

AFRL-VA-WP-TR-2005-3041

**HIGH CONFIDENCE
RECONFIGURABLE DISTRIBUTED
CONTROL**

**Jason Hickey
John Hauser
Richard Murray**

**California Institute of Technology
Office of Sponsored Research
1201 E. California Blvd.
Pasadena, CA 91125**



APRIL 2005

Final Report for 25 September 1998 – 28 September 2004

Approved for public release; distribution is unlimited.

STINFO FINAL REPORT

**AIR VEHICLES DIRECTORATE
AIR FORCE MATERIEL COMMAND
AIR FORCE RESEARCH LABORATORY
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7542**

NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Air Force Research Laboratory Wright Site Public Affairs Office (AFRL/WS) and is releasable to the National Technical Information Service (NTIS). It will be available to the general public, including foreign nationals.

THIS TECHNICAL REPORT IS APPROVED FOR PUBLICATION.

/s/

COREY SCHUMACHER
Program Manager
Control Design and Analysis Branch

/s/

DEBORAH S. GRISMER
Chief, Control Design and Analysis Branch
Air Force Research Laboratory

/s/

BRIAN W. VAN VLIET, Chief
Control Sciences Division
Air Vehicles Directorate

This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>						
1. REPORT DATE (DD-MM-YY) April 2005		2. REPORT TYPE Final		3. DATES COVERED (From - To) 09/25/1998 – 09/28/2004		
4. TITLE AND SUBTITLE HIGH CONFIDENCE RECONFIGURABLE DISTRIBUTED CONTROL				5a. CONTRACT NUMBER F33615-98-C-3613		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER 0602302		
6. AUTHOR(S) Jason Hickey John Hauser Richard Murray				5d. PROJECT NUMBER A04H		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER 0D		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) California Institute of Technology Office of Sponsored Research 1201 E. California Blvd. Pasadena, CA 91125				8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Vehicles Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson Air Force Base, OH 45433-7542				10. SPONSORING/MONITORING AGENCY ACRONYM(S) AFRL/VACA		
				11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S) AFRL-VA-WP-TR-2005-3041		
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES Report contains color.						
14. ABSTRACT The Caltech/Colorado SEC project developed and tested two major advances in software enabled control: optimization-based control using real-time trajectory generation and logical programming environments for formal analysis of distributed control systems. These two activities, funded under the OCC and HSCC tasks of the SEC, were integrated and tested on the industry-led demonstration using the F-15 and T-33 flight tests.						
15. SUBJECT TERMS Software Enabled Control, SEC, reconfigurable control, UAV						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT: SAR	18. NUMBER OF PAGES 48	19a. NAME OF RESPONSIBLE PERSON (Monitor) Corey Schumacher 19b. TELEPHONE NUMBER (Include Area Code) (937) 255-8682	
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified				

High Confidence Reconfigurable Distributed Control

Jason Hickey John Hauser Richard Murray (PI)
California Institute of Technology/University of Colorado

F33615-98-C-3613

Abstract

The Caltech/Colorado SEC project developed and tested two major advances in software enabled control: optimization-based control using real-time trajectory generation and logical programming environments for formal analysis of distributed control systems. These two activities, funded under the OCC and HSCC tasks of the SEC, were integrated and tested on the industry-led demonstration using the F-15 and T-33 flight tests.

1 Project Overview

The theme of this SEC project was the development of distributed control systems that can be dynamically re-configured and in which DoD can have high confidence. In this project we are developing the underlying theory, building software tools, and experimentally implementing our results. The specific objectives of this program were:

- To work towards a general methodology, including theory, languages and automated support, for developing software enabled control systems. This includes real-time, optimization-based control theory and "correct by construction" programming techniques, as well as tools for building systems for specific purposes.
- To develop the theory, algorithms, and designs to address specific problems in SEC, including multi-vehicle, hierarchical, distributed control for cooperative tasks, and "on-the-fly" techniques for adapting to changes and strategy in dynamic distributed environments.
- To demonstrate the feasibility of our methodology by implementing it using the open control platform (OCP) on an experimental platform consisting of a single vehicle performing aggressive maneuvers with uncertainty and failures, and multiple vehicles performing a coordinated task in an adversarial environment.
- The work that we envision will focus on the theoretical aspects of the problem, combined with software tools required to implement our approach. A proof of concept implementation will be used to demonstrate the key ideas and help sharpen the tools that are developed.
- The program will build on the receding-horizon, model predictive control paradigm that has been developed under the current SEC program. This approach allows automatic reconfiguration of the control law based on changes in condition of the vehicle, the external environment, and the mission specification. Theoretical work has shown that by combining receding horizon control with terminal costs based on control Lyapunov functions, a computational tractable

approach to real-time optimization can be developed. We are in the process of implementing this work on the Caltech ducted fan and will expand the approach to allow trajectory tracking, multi-vehicle operation, and robustness guarantees.

As part of our activities, we have developed a software environment for design of real-time, control systems that generates validated, embedded software for use as part of the open control platform. This environment includes languages for specifying desired performance of the system, translation tools for converting these performance specifications into control specifications in a semi-automated fashion, and verification tools that allow the (hybrid) control algorithms to be validated against the specification in a way that describes the conditions required for successful execution. This environment has been built on top of the existing software tools that were developed at Caltech for real-time, optimization-based control, as well as the open control platform. A novel feature of the software environment is the ability to support dynamic reconfiguration to accommodate changes in mission, condition, and external environment.

2 OCC: Optimization-Based Control

An advanced control technology that has had tremendous practical impact over the last two decades or so is model predictive control (MPC). By embedding an optimization solution in each sampling instant within the control calculation, MPC applications have demonstrated dramatic improvements in control performance. To date, this impact has largely been limited to the process industries. The reasons for the domain-specific benefit have to do with the relatively slow time constants of most industrial processes and their relatively benign dynamics (e.g., their open-loop stability). For aerospace systems to avail of the promise of MPC, research is needed in extending the technology so that it can be applied to systems with nonlinear, unstable, and fast dynamics.

Under the SEC program, we built a new framework for MPC and optimization-based control for flight control applications [7, 6, 2, 11, 4, 5, 3]. The MPC formulation replaces the traditional terminal constraint with a terminal cost based on a control Lyapunov function. This reduces computational requirements and allows proofs of stability under a variety of realistic assumptions on computation.

A major element of our approach was to use differential flatness system to gain computational advantage [12]. (A system is differentially flat if, roughly, it can be modeled as a dynamical equation in one variable and its derivatives.) In this case the optimization can be done over a space of parametrized basis functions and a constrained nonlinear program can be solved using collocation points. A software package, NTG, was been developed to implement these theoretical developments and this package was transitioned to industry (as part of the Northrop Grumman SEC flight demonstration).

To test these algorithms, a tethered ducted fan testbed was developed at Caltech that mimics the longitudinal dynamics of an aircraft [1, 2, 5]. High-performance maneuvers can safely be flown with the ducted fan and an interface to high-end workstations allows complex control schemes to be solved in real-time. Optimization-based control using NTG has been implemented in several experiments, detailing among other things the effects of different MPC optimization horizons on computing time and dynamic performance.

More information on this topic is included in Appendix B.

3 HCSS: Computation and Control Language

A cooperative control system consists of multiple, autonomous components interacting to control their environment. Examples include air traffic control systems, automated factories, robot soccer teams and sensor/actuator networks. In each of these systems, a component reacts to its environment and to messages received from neighboring components. Thus, a cooperative control system is at once a controlled physical system and a distributed computer. Designing cooperative control systems, therefore, requires a combination of tools from control theory and distributed systems.

Under the SEC program, we developed a language for describing and reasoning about such systems, called CCL (Computational and Control Language) [8, 10, 9, 13, 14]. CCL is an attempt to bridge the ways of modeling and designing software and dynamical systems. It is designed for use in distributed (partially asynchronous) environments, such as those required for multi-vehicle operations.

CCL is a modeling language similar in appearance to UNITY, but interpreted differently. The basic unit of a CCL program is the guarded command (or simply command) which we describe by example. Formal definitions can be found elsewhere [9]. An example of a guarded command is:

$$t > 10 : x' \geq x + 1 \wedge t' = 0 :$$

The part before the colon is called the guard and the part after it is called the rule. We interpret it as follows: If this command is executed in a state where the variable t is greater than 10, then a new state will result in which the new value of x is greater than or equal to its old value plus 1, and the new value of t is 0. All other variables (those not occurring primed) remain the same. If the command is executed in a state in which t is not greater than 10, then the new state is defined to be exactly the same as the old state. The execution of a command is called a step. Note that guarded commands can be non-deterministic, as is the one above since it does not specify the exact new value for x , only that it should increase by at least 1.

A complete CCL program $P = (I, C)$ consists of two parts: An initial predicate I that says what the initial values of the variables involved are allowed to be; and a set C of guarded commands. CCL program composition is very straightforward. If $P_1 = (I_1, C_1)$ and $P_2 = (I_2, C_2)$, then their composition is simply $P_1 \circ P_2 = (I_1 \wedge I_2, C_1 \cup C_2)$. That is, to obtain the composition of two programs, conjoin their initial clauses and union their command sets.

More information on this topic is included in Appendix C.

4 Final Demo

The Caltech/Colorado team participated in the industry-led demo by implementing a reliable wingman scenario, in which a UAV performed formation flying with a manned aircraft, demonstrating both performance and safety, and operated in the presence of communications failures by executing a lost wingman scenario. During formation flight, the vehicles performed a set of coordinated actions using receding horizon control with on the UAV to demonstrate real-time trajectory generation. After demonstrating coordinated formation flight, the UAV simulated loss of the high data rate link between the aircraft and begin executing a “lost wingman” protocol designed to (provably) achieve an increased separation between the aircraft. After safe separation was achieved, the high bandwidth link was restored and the UAV requested a rejoin and executed a safe rejoin maneuver.

The Logical Programming Environment (LPE) framework was used to develop and verify the protocols for executing this scenario, ensuring that the problem specification was satisfied in pres-

ence of uncertainty in dynamics, timing and failures. Temporal logic was used to specify the desired performance and our protocol was implemented using CCL (Computation and Control Language).

More information on this topic is included in Appendix D.

5 Conclusions

The Caltech/Colorado project within the SEC program generated two major advances in control theory and practice: a framework for optimization-based control using NTG and a logical programming environment built around CCL. Together, these two advances have provided new insights, approaches and tools for real-time, cooperative control that exploits advances in computing and communications.

References

- [1] W. B. Dunbar, M. B. Milam, R. Franz, and R. M. Murray. Model predictive control of a thrust-vectored flight control experiment. In *Proc. IFAC World Congress*, 2002.
- [2] R. Franz, M. B. Milam, and J. E. Hauser. Applied receding horizon control of the caltech ducted fan. In *Proc. American Control Conference*, 2002. Submitted.
- [3] A. Jadbabaie. *Nonlinear Receding Horizon Control: A Control Lyapunov Function Approach*. PhD thesis, California Institute of Technology, Control and Dynamical Systems, 2001.
- [4] A. Jadbabaie and J. E. Hauser. Relaxing the optimality condition in receding horizon control. In *Proc. IEEE Control and Decision Conference*, 2000.
- [5] A. Jadbabaie, J. Yu, and J. E. Hauser. Receding horizon control of the caltech ducted fan: A control Lyapunov function approach. In *Proc. IEEE International Conference on Control and Applications*, 1999.
- [6] A. Jadbabaie, J. Yu, and J. E. Hauser. The region of attraction of receding horizon control strategies with control Lyapunov functions. In *Proc. IEEE Control and Decision Conference*, 1999.
- [7] A. Jadbabaie, J. Yu, and J. E. Hauser. Stabilizing receding horizon control of nonlinear systems: A control Lyapunov function approach. In *Proc. American Control Conference*, 1999.
- [8] E. Klavins. Communication complexity of multi-robot systems. In *Fifth International Workshop on the Algorithmic Foundations of Robotics*, 2002.
- [9] E. Klavins. A formal model of a multi-robot control and communication task. In *Proc. IEEE Control and Decision Conference*, 2003.
- [10] E. Klavins and R. M. Murray. Distributed computation for cooperative control. *IEEE Pervasive Computing*, 2003. In revision.
- [11] M. B. Milam, R. Franz, J. E. Hauser, and R. M. Murray. Receding horizon control of a vectored thrust flight experiment. *IEE Proceedings on Control Theory and Applications*, 2003. Submitted.
- [12] R. M. Murray *et al.* Online control customization via optimization-based control. In T. Samad and G. Balas, editors, *Software-Enabled Control: Information Technology for Dynamical Systems*. IEEE Press, 2003.
- [13] D. Del Vecchio and E. Klavins. Observation of hybrid guarded command programs. In *Proc. IEEE Control and Decision Conference*, 2003. To appear.
- [14] D. Del Vecchio, R. M. Murray, and Erik Klavins. Discrete state estimators for systems on a lattice. *Automatica*, 2004. Submitted.

A Individuals Supported by Contract

Faculty

- Mani Chandy (Caltech CS; 1998-2000)
- John Doyle (Caltech CDS; 1998-2000)
- Jason Hickey (Caltech CS; 2001–2004)
- John Hauser (Colorado EE; 2000–2004)
- Richard Murray (PI; Caltech CDS; 1998–2004)

Postdoctoral Scholars

- Eric Klavins (Caltech CS, 2001–2003). Currently an assistant professor of electrical engineering at the University of Washington.
- Reza Olfati-Saber (Caltech CDS, 2001–2004). Currently an instructor in Mechanical and Aerospace Engineering at UCLA.
- Nicolas Petit (Caltech CDS, 2000–2002). Currently an assistant professor at Ecoles des Mines des Paris.

Graduate Students

- Bill Dunbar (Caltech CDS) - Currently an assistant professor of electrical and computer engineering at UC Santa Cruz.
- Melvin Flores (Caltech CDS)
- Ryan Franz (Colorado EE) - Currently employed by Northrop Grumman Corporation.
- Roman Ginis (Caltech CS)
- Adam Granicz (Caltech CS)
- Nathan Gray (Caltech CS)
- Rick Hindman (Colorado EE) - Currently employed by Raytheon. Corporation
- Ali Jadbabaie (Caltech CDS) - Currently an assistant professor of electrical engineering at the University of Pennsylvania.
- Mark Milam (Caltech CDS) - Currently employed by Northrop Grumman Corporation.
- Cristian Tapus (Caltech CS)
- Steve Waydo (Caltech CDS)

Online Control Customization via Optimization-Based Control*

Richard M. Murray

Control and Dynamical Systems
California Institute of Technology
Pasadena, CA 91125

John Hauser

Electrical and Computer Engineering
University of Colorado
Boulder, CO 80309

Ali Jadbabaie[†], Mark B. Milam, Nicolas Petit[‡], William B. Dunbar, Ryan Franz[§]

Control and Dynamical Systems
California Institute of Technology

Submitted, *Software-Enabled Control: Information Technology for Dynamical Systems*, T. Samad and G. Balas (eds), IEEE Press, 2002 (in preparation)

Online Control Customization via Optimization-Based Control

Editors' Summary

An advanced control technology that has had tremendous practical impact over the last two decades or so is model predictive control (MPC). By embedding an optimization solution in each sampling instant within the control calculation, MPC applications have demonstrated dramatic improvements in control performance. To date, this impact has largely been limited to the process industries. The reasons for the domain-specific benefit have to do with the relatively slow time constants of most industrial processes and their relatively benign dynamics (e.g., their open-loop stability). For aerospace systems to avail of the promise of MPC, research is needed in extending the technology so that it can be applied to systems with nonlinear, unstable, and fast dynamics.

This chapter presents a new framework for MPC and optimization-based control for flight control applications. The MPC formulation replaces the traditional terminal constraint with a terminal cost based on a control Lyapunov function. This reduces computational requirements and allows proofs of stability under a variety of realistic assumptions on computation.

The authors also show how differential flatness system can be used to computational advantage. (A system is differentially flat if, roughly, it can be modeled as a dynamical equation in one variable and its derivatives.) In this case the optimization can be done over a space of parametrized basis functions and a constrained nonlinear program can be solved using collocation points. A software package has been developed to implement these theoretical developments.

There is an experimental component to this research as well. A tethered ducted fan testbed has been developed at Caltech that mimics the longitudinal dynamics of an aircraft. High-performance maneuvers can safely be flown with the ducted fan and an interface to high-end workstations

*Research supported in part by DARPA contract F33615-98-C-3613.

[†]Currently with the Department of Electrical Engineering, Yale University

[‡]Currently with Centre Automatique et Systèmes, École National Supérieure des Mines, Paris

[§]Electrical and Computer Engineering, University of Colorado

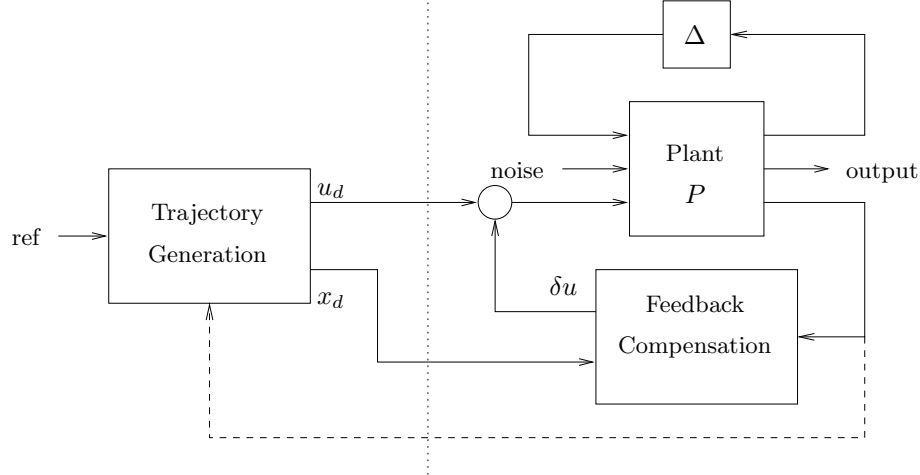


Figure 1: Two degree of freedom controller design for a plant P with uncertainty Δ . See text for a detailed explanation.

allows complex control schemes to be solved in real-time. The chapter presents results from several experiments, detailing among other things the effects of different MPC optimization horizons on computing time and dynamic performance.

Another model predictive control approach is discussed in Chapter 10.

1 Introduction

A large class of industrial and military control problems consist of planning and following a trajectory in the presence of noise and uncertainty. Examples include unmanned airplanes and submarines for surveillance and combat, mobile robots in factories and on the surface of Mars, and medical robots performing inspection and manipulation tasks inside the human body under the control of a surgeon. All of these systems are highly nonlinear and demand accurate performance.

To control such systems, we make use of the notion of *two degree of freedom* controller design. This is a standard technique in linear control theory that separates a controller into a feedforward compensator and a feedback compensator. The feedforward compensator generates the nominal input required to track a given reference trajectory. The feedback compensator corrects for errors between the desired and actual trajectories. This is shown schematically in Figure 1.

In a nonlinear setting, two degree of freedom controller design decouples the trajectory generation and asymptotic tracking problems. Given a desired output trajectory, we first construct a state space trajectory x_d and a nominal input u_d that satisfy the equations of motion. The error system can then be written as a time-varying control system in terms of the error, $e = x - x_d$. Under the assumption that that tracking error remains small, we can linearize this time-varying system about $e = 0$ and stabilize the $e = 0$ state. A more detailed description of this approach, including references to some of the related literature, is given in [24].

In optimization-based control, we use the two degree of freedom paradigm with an optimal control computation for generating the feasible trajectory. In addition, we take the extra step of updating the generated trajectory based on the current state of the system. This additional feedback path is denoted by a dashed line in Figure 1 and allows the use of so-called *receding horizon* control techniques: a (optimal) feasible trajectory is computed from the current position

to the desired position over a finite time T horizon, used for a short period of time $\delta < T$, and then recomputed based on the new position.

Many variations on this approach are possible, blurring the line between the trajectory generation block and the feedback compensation. For example, if $\delta \ll T$, one can eliminate all or part of the “inner loop” feedback compensator, relying on the receding horizon optimization to stabilize the system. A local feedback compensator may still be employed to correct for errors due to noise and uncertainty on the fastest time scales. In this chapter, we will explore both the case where we have a relatively large δ , in which case we consider the problem to be primarily one of trajectory generation, and a relatively small δ , where optimization is used for stabilizing the system.

A key advantage of optimization-based approaches is that they allow the potential for customization of the controller based on changes in *mission*, *condition*, and *environment*. Because the controller is solving the optimization problem online, updates can be made to the cost function, to change the desired operation of the system; to the model, to reflect changes in parameter values or damage to sensors and actuators; and to the constraints, to reflect new regions of the state space that must be avoided due to external influences. Thus, many of the challenges of designing controllers that are robust to a large set of possible uncertainties become embedded in the online optimization.

Development and application of receding horizon control (also called model predictive control, or MPC) originated in process control industries where plants being controlled are sufficiently slow to permit its implementation. An overview of the evolution of commercially available MPC technology is given in [27] and a survey of the current state of stability theory of MPC is given in [20]. Closely related to the work in this chapter, Singh and Fuller [29] have used MPC to stabilize a linearized simplified UAV helicopter model around an open-loop trajectory, while respecting state and input constraints.

In the remainder of this chapter, we give a survey of the tools required to implement online control customization via optimization-based control. Section 2 introduces some of the mathematical results and notation required for the remainder of the chapter. Section 3 gives the main theoretical results of the paper, where the problem of receding horizon control using a control Lyapunov function (CLF) as a terminal cost is described. In Section 4, we provide a computational framework for computing optimal trajectories in real-time, a necessary step toward implementation of optimization-based control in many applications. Finally, in Section 5, we present an experimental implementation of both real-time trajectory generation and model-predictive control on a flight control experiment.

The results in this chapter are based in part on work presented elsewhere. The work on receding horizon control using a CLF terminal cost was developed by Jadbabaie, Hauser and co-workers and is described in [16]. The real-time trajectory generation framework, and the corresponding software, was developed by Milam and co-workers and has appeared in [21, 23, 25]. The implementation of model predictive control given in this chapter is based on the work of Dunbar, Milam, and Franz [8, 10].

2 Mathematical Preliminaries

In this section we provide some mathematical preliminaries and establish the notation used through the chapter. We consider a nonlinear control system of the form

$$\dot{x} = f(x, u) \tag{1}$$

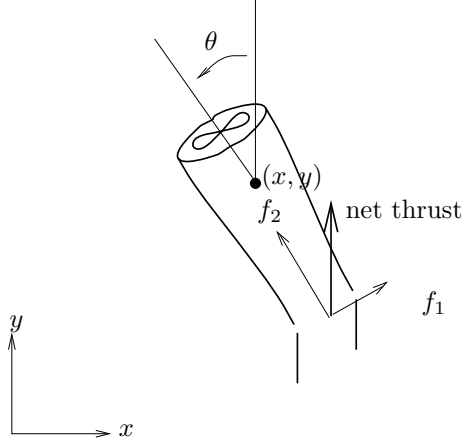


Figure 2: Planar ducted fan engine. Thrust is vectored by moving the flaps at the end of the duct.

where the vector field $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ is at least C^2 and possesses a linearly controllable equilibrium point at the origin, e.g., $f(0, 0) = 0$ and $(A, B) := (D_1 f(0, 0), D_2 f(0, 0))$ is controllable.

2.1 Differential Flatness and Trajectory Generation

For optimization-based control in applications such as flight control, a critical need is the ability to compute optimal trajectories *very* quickly, so that they can be used in a real-time setting. For general problems this can be very difficult, but there are classes of systems for which simplifications can be made that vastly reduce the computational requirements for generating trajectories. We describe one such class of systems here, so-called differentially flat systems.

Roughly speaking, a system is said to be differentially flat if all of the feasible trajectories for the system can be written as functions of a flat output $z(\cdot)$ and its derivatives. More precisely, given a nonlinear control system (1) we say the system is *differentially flat* if there exists a function $z(x, u, \dot{u}, \dots, u^{(p)})$ such that all feasible solutions of the differential equation (1) can be written as

$$\begin{aligned} x &= \alpha(z, \dot{z}, \dots, z^{(q)}) \\ u &= \beta(z, \dot{z}, \dots, z^{(q)}). \end{aligned} \tag{2}$$

Differentially flat systems were originally studied by Fliess et al. [9]. See [24] for a description of the role of flatness in control of mechanical systems and [33] for more information on flatness applied to flight control systems.

Example 1 (Planar ducted fan). Consider the dynamics of a planar, vectored thrust flight control system as shown in Figure 2. This system consists of a rigid body with body fixed forces and is a simplified model for the Caltech ducted fan described in Section 5. Let (x, y, θ) denote the position and orientation of the center of mass of the fan. We assume that the forces acting on the fan consist of a force f_1 perpendicular to the axis of the fan acting at a distance r from the center of mass, and a force f_2 parallel to the axis of the fan. Let m be the mass of the fan, J the moment of inertia, and g the gravitational constant. We ignore aerodynamic forces for the purpose of this example.

The dynamics for the system are

$$\begin{aligned} m\ddot{x} &= f_1 \cos \theta - f_2 \sin \theta \\ m\ddot{y} &= f_1 \sin \theta + f_2 \cos \theta - mg \\ J\ddot{\theta} &= rf_1. \end{aligned} \tag{3}$$

Martin et al. [19] showed that this system is differentially flat and that one set of flat outputs is given by

$$\begin{aligned} z_1 &= x - (J/mr) \sin \theta \\ z_2 &= y + (J/mr) \cos \theta. \end{aligned} \tag{4}$$

Using the system dynamics, it can be shown that

$$\ddot{z}_1 \cos \theta + (\ddot{z}_2 + g) \sin \theta = 0. \tag{5}$$

and thus given $z_1(t)$ and $z_2(t)$ we can find $\theta(t)$ except for an ambiguity of π and away from the singularity $\ddot{z}_1 = \ddot{z}_2 + g = 0$. The remaining states and the forces $f_1(t)$ and $f_2(t)$ can then be obtained from the dynamic equations, all in terms of z_1 , z_2 , and their higher order derivatives.

Having determined that a system is flat, it follows that all feasible trajectories for the system are characterized by the evolution of the flat outputs. Using this fact, we can convert the problem of point to point motion generation to one of finding a curve $z(\cdot)$ which joins an initial state $z(0), \dot{z}(0), \dots, \dot{z}^{(q)}(0)$ to a final state. In this way, we reduce the problem of generating a feasible trajectory for the system to a classical algebraic problem in interpolation. Similarly, problems in generation of trajectories to track a reference signal can also be converted to problems involving curves $z(\cdot)$ and algebraic methods can be used to provide real-time solutions [33, 34].

Thus, for differentially flat systems, trajectory generation can be reduced from a dynamic problem to an algebraic one. Specifically, one can parameterize the flat outputs using basis functions $\phi_i(t)$,

$$z = \sum a_i \phi_i(t), \tag{6}$$

and then write the feasible trajectories as functions of the coefficients a :

$$\begin{aligned} x_d &= \alpha(z, \dot{z}, \dots, z^{(q)}) = x_d(a) \\ u_d &= \beta(z, \dot{z}, \dots, z^{(q)}) = u_d(a). \end{aligned} \tag{7}$$

Note that no ODEs need to be integrated in order to compute the feasible trajectories (unlike optimal control methods, which involve parameterizing the *input* and then solving the ODEs). This is the defining feature of differentially flat systems. The practical implication is that nominal trajectories and inputs which satisfy the equations of motion for a differentially flat system can be computed in a computationally efficient way (solution of algebraic equations).

2.2 Control Lyapunov Functions

For the optimal control problems that we introduce in the next section, we will make use of a terminal cost that is also a control Lyapunov function for the system. Control Lyapunov functions are an extension of standard Lyapunov functions and were originally introduced by Sontag [30]. They allow constructive design of nonlinear controllers and the Lyapunov function that proves their stability. A more complete treatment is given in [17].

Definition 1. Control Lyapunov Function

A locally positive function $V : \mathbb{R}^n \rightarrow \mathbb{R}_+$ is called a *control Lyapunov function (CLF)* for a control system (1) if

$$\inf_{u \in \mathbb{R}^m} \left(\frac{\partial V}{\partial x} f(x, u) \right) < 0 \quad \text{for all } x \neq 0.$$

In general, it is difficult to find a CLF for a given system. However, for many classes of systems, there are specialized methods that can be used. One of the simplest is to use the Jacobian linearization of the system around the desired equilibrium point and generate a CLF by solving an LQR problem.

It is a well known result that the problem of minimizing the quadratic performance index,

$$J = \int_0^\infty (x^T(t)Qx(t) + u^T Ru(t))dt \quad \text{subject to} \quad \dot{x} = Ax + Bu, \quad x(0) = x_0, \quad (8)$$

results in finding the positive definite solution of the following Riccati equation:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \quad (9)$$

The optimal control action is given by

$$u = -R^{-1}B^T Px$$

and $V = x^T Px$ is a CLF for the system.

In the case of the nonlinear system $\dot{x} = f(x, u)$, A and B are taken as

$$A = \frac{\partial f(x, u)}{\partial x} \Big|_{(0,0)} \quad B = \frac{\partial f(x, u)}{\partial u} \Big|_{(0,0)}$$

where the pairs (A, B) and $(Q^{\frac{1}{2}}, A)$ are assumed to be stabilizable and detectable respectively. Obviously the obtained CLF $V(x) = x^T Px$ will be valid only in a region around the equilibrium $(0, 0)$.

More complicated methods for finding control Lyapunov functions are often required and many techniques have been developed. An overview of some of these methods can be found in [15].

3 Optimization-Based Control

In receding horizon control, a finite horizon optimal control problem is solved, generating an open-loop state and control trajectories. The resulting control trajectory is then applied to the system for a fraction of the horizon length. This process is then repeated, resulting in a sampled data feedback law. Although receding horizon control has been successfully used in the process control industry, its application to fast, stability critical nonlinear systems has been more difficult. This is mainly due to two issues. The first is that the finite horizon optimizations must be solved in a relatively short period of time. Second, it can be demonstrated using linear examples that a naive application of the receding horizon strategy can have disastrous effects, often rendering a system unstable. Various approaches have been proposed to tackle this second problem; see [20] for a comprehensive review of this literature. The theoretical framework presented here also addresses the stability issue directly, but is motivated by the need to relax the computational demands of existing stabilizing MPC formulations.

A number of approaches in receding horizon control employ the use of terminal state equality or inequality constraints, often together with a terminal cost, to ensure closed loop stability. In Primbs et al. [26], aspects of a stability-guaranteeing, global control Lyapunov function were used, via state and control constraints, to develop a stabilizing receding horizon scheme. Many of the nice characteristics of the CLF controller together with better cost performance were realized. Unfortunately, a global control Lyapunov function is rarely available and often not possible.

Motivated by the difficulties in solving constrained optimal control problems, we have developed an alternative receding horizon control strategy for the stabilization of nonlinear systems [16]. In this approach, closed loop stability is ensured through the use of a terminal cost consisting of a control Lyapunov function that is an incremental upper bound on the optimal cost to go. This terminal cost eliminates the need for terminal constraints in the optimization and gives a dramatic speed-up in computation. Also, questions of existence and regularity of optimal solutions (very important for online optimization) can be dealt with in a rather straightforward manner. In the remainder of this section, we review the results presented in [16].

3.1 Finite Horizon Optimal Control

We first consider the problem of optimal control over a finite time horizon. Given an initial state x_0 and a control trajectory $u(\cdot)$ for a nonlinear control system $\dot{x} = f(x, u)$, the state trajectory $x^u(\cdot; x_0)$ is the (absolutely continuous) curve in \mathbb{R}^n satisfying

$$x^u(t; x_0) = x_0 + \int_0^t f(x^u(\tau; x_0), u(\tau)) d\tau$$

for $t \geq 0$.

The performance of the system will be measured by a given incremental cost $q : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ that is C^2 and fully penalizes both state and control according to

$$q(x, u) \geq c_q(\|x\|^2 + \|u\|^2), \quad x \in \mathbb{R}^n, u \in \mathbb{R}^m$$

for some $c_q > 0$ and $q(0, 0) = 0$. It follows that the quadratic approximation of q at the origin is positive definite, $D^2q(0, 0) \geq c_q I > 0$.

To ensure that the solutions of the optimization problems of interest are well behaved, we impose some convexity conditions. We require the set $f(x, \mathbb{R}^m) \subset \mathbb{R}^n$ to be convex for each $x \in \mathbb{R}^n$. Letting $p \in \mathbb{R}^n$ represent the co-state, we also require that the pre-Hamiltonian function $u \mapsto p^T f(x, u) + q(x, u) =: K(x, u, p)$ be strictly convex for each $(x, p) \in \mathbb{R}^n \times \mathbb{R}^n$ and that there is a C^2 function $\bar{u}^* : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^m : (x, p) \mapsto \bar{u}^*(x, p)$ providing the global minimum of $K(x, u, p)$. The Hamiltonian $H(x, p) := K(x, \bar{u}^*(x, p), p)$ is then C^2 , ensuring that extremal state, co-state, and control trajectories will all be sufficiently smooth (C^1 or better). Note that these conditions are trivially satisfied for control affine f and quadratic q .

The cost of applying a control $u(\cdot)$ from an initial state x over the infinite time interval $[0, \infty)$ is given by

$$J_\infty(x, u(\cdot)) = \int_0^\infty q(x^u(\tau; x), u(\tau)) d\tau.$$

The optimal cost (from x) is given by

$$J_\infty^*(x) = \inf_{u(\cdot)} J_\infty(x, u(\cdot))$$

where the control functions $u(\cdot)$ belong to some reasonable class of admissible controls (e.g., piecewise continuous or measurable). The function $x \mapsto J_\infty^*(x)$ is often called the *optimal value function* for the infinite horizon optimal control problem.

For the class of f and q considered, we know that $J_\infty^*(\cdot)$ is a positive definite C^2 function on a neighborhood of the origin. This follows from the *geometry* of the corresponding Hamiltonian system (see [13] and the references therein). In particular, since $(x, p) = (0, 0)$ is a hyperbolic critical point of the C^1 Hamiltonian vector field $X_H(x, p) := (D_2 H(x, p), -D_1 H(x, p))^T$, the local properties of $J_\infty^*(\cdot)$ are determined by the linear-quadratic approximation to the problem and, moreover, $D^2 J_\infty^*(0) = P > 0$ where P is the stabilizing solution of the appropriate algebraic Riccati equation.

For practical purposes, we are interested in finite horizon approximations of the infinite horizon optimization problem. In particular, let $V(\cdot)$ be a nonnegative C^2 function with $V(0) = 0$ and define the finite horizon cost (from x using $u(\cdot)$) to be

$$J_T(x, u(\cdot)) = \int_0^T q(x^u(\tau; x), u(\tau)) d\tau + V(x^u(T; x)) \quad (10)$$

and denote the optimal cost (from x) as

$$J_T^*(x) = \inf_{u(\cdot)} J_T(x, u(\cdot)) .$$

As in the infinite horizon case, one can show, by geometric means, that $J_T^*(\cdot)$ is locally smooth (C^2). Other properties will depend on the choice of V and T .

Let Γ^∞ denote the domain of $J_\infty^*(\cdot)$ (the subset of \mathbb{R}^n on which J_∞^* is finite). It is not too difficult to show that the cost functions $J_\infty^*(\cdot)$ and $J_T^*(\cdot)$, $T \geq 0$, are continuous functions on Γ^∞ [15]. For simplicity, we will allow $J_\infty^*(\cdot)$ to take values in the extended real line so that, for instance, $J_\infty^*(x) = +\infty$ means that there is no control taking x to the origin.

We will assume that f and q are such that the minimum value of the cost functions $J_\infty^*(x)$, $J_T^*(x)$, $T \geq 0$, is attained for each (suitable) x . That is, given x and $T > 0$ (including $T = \infty$ when $x \in \Gamma^\infty$), there is a (C^1 in t) optimal trajectory $(x_T^*(t; x), u_T^*(t; x))$, $t \in [0, T]$, such that $J_T(x, u_T^*(\cdot; x)) = J_T^*(x)$. For instance, if f is such that its trajectories can be bounded on finite intervals as a function of its input size, e.g., there is a continuous function β such that $\|x^u(t; x_0)\| \leq \beta(\|x_0\|, \|u(\cdot)\|_{L_1[0, t]})$, then (together with the conditions above) there will be a minimizing control (cf. [18]). Many such conditions may be used to good effect; see [15] for a more complete discussion.

It is easy to see that $J_\infty^*(\cdot)$ is proper on its domain so that the sub-level sets

$$\Gamma_r^\infty := \{x \in \Gamma^\infty : J_\infty^*(x) \leq r^2\}$$

are compact and path connected and moreover $\Gamma^\infty = \bigcup_{r \geq 0} \Gamma_r^\infty$. Note also that Γ^∞ may be a proper subset of \mathbb{R}^n since there may be states that cannot be driven to the origin. We use r^2 (rather than r) here to reflect the fact that our incremental cost is quadratically bounded from below. We refer to sub-level sets of $J_T^*(\cdot)$ and $V(\cdot)$ using

$$\Gamma_r^T := \text{path connected component of } \{x \in \Gamma^\infty : J_T^*(x) \leq r^2\} \text{ containing } 0,$$

and

$$\Omega_r := \text{path connected component of } \{x \in \mathbb{R}^n : V(x) \leq r^2\} \text{ containing } 0.$$

These results provide the technical framework needed for receding horizon control.

3.2 Receding Horizon Control with CLF Terminal Cost

Receding horizon control provides a practical strategy for the use of model information through on-line optimization. Every δ seconds, an optimal control problem is solved over a T second horizon, starting from the current state. The first δ seconds of the optimal control $u_T^*(\cdot; x(t))$ is then applied to the system, driving the system from $x(t)$ at current time t to $x_T^*(\delta, x(t))$ at the next sample time $t + \delta$ (assuming no model uncertainty). We denote this receding horizon scheme as $\mathcal{RH}(T, \delta)$.

In defining (unconstrained) finite horizon approximations to the infinite horizon problem, the key design parameters are the terminal cost function $V(\cdot)$ and the horizon length T (and, perhaps also, the increment δ). What choices will result in success?

It is well known (and easily demonstrated with linear examples), that simple truncation of the integral (i.e., $V(x) \equiv 0$) may have disastrous effects if $T > 0$ is too small. Indeed, although the resulting value function may be nicely behaved, the “optimal” receding horizon closed loop system can be unstable.

A more sophisticated approach is to make good use of a suitable terminal cost $V(\cdot)$. Evidently, the best choice for the terminal cost is $V(x) = J_\infty^*(x)$ since then the optimal finite and infinite horizon costs are the same. Of course, if *the* optimal value function were available there would be no need to solve a trajectory optimization problem. What properties of the optimal value function should be retained in the terminal cost? To be effective, the terminal cost should account for the discarded tail by ensuring that the origin can be reached from the terminal state $x^u(T; x)$ in an efficient manner (as measured by q). One way to do this is to use an appropriate control Lyapunov function which is also an upper bound on the cost-to-go.

The following theorem shows that the use of a particular type of CLF is in fact effective, providing rather strong and specific guarantees.

Theorem 1. [16] *Suppose that the terminal cost $V(\cdot)$ is a control Lyapunov function such that*

$$\min_{u \in \mathbb{R}^m} (\dot{V} + q)(x, u) \leq 0 \quad (11)$$

for each $x \in \Omega_{r_v}$ for some $r_v > 0$. Then, for every $T > 0$ and $\delta \in (0, T]$, the resulting receding horizon trajectories go to zero exponentially fast. For each $T > 0$, there is an $\bar{r}(T) \geq r_v$ such that $\Gamma_{\bar{r}(T)}^T$ is contained in the region of attraction of $\mathcal{RH}(T, \delta)$. Moreover, given any compact subset Λ of Γ^∞ , there is a T^ such that $\Lambda \subset \Gamma_{\bar{r}(T)}^T$ for all $T \geq T^*$.*

Theorem 1 shows that for *any* horizon length $T > 0$ and *any* sampling time $\delta \in (0, T]$, the receding horizon scheme is exponentially stabilizing over the set $\Gamma_{r_v}^T$. For a given T , the region of attraction estimate is enlarged by increasing r beyond r_v to $\bar{r}(T)$ according to the requirement that $V(x_T^*(T; x)) \leq r_v^2$ on that set. An important feature of the above result is that, for operations with the set $\Gamma_{\bar{r}(T)}^T$, there is no need to impose stability ensuring constraints which would likely make the online optimizations more difficult and time consuming to solve.

An important benefit of receding horizon control is its ability to handle state and control constraints. While the above theorem provides stability guarantees when there are no constraints present, it can be modified to include constraints on states and controls as well. In order to ensure stability when state and control constraints are present, the terminal cost $V(\cdot)$ should be a local CLF satisfying $\min_{u \in \mathcal{U}} \dot{V} + q(x, u) \leq 0$ where \mathcal{U} is the set of controls where the control constraints are satisfied. Moreover, one should also require that the resulting state trajectory $x^{CLF}(\cdot) \in \mathcal{X}$, where \mathcal{X} is the set of states where the constraints are satisfied. (Both \mathcal{X} and \mathcal{U} are assumed to be compact with origin in their interior). Of course, the set Ω_{r_v} will end up being smaller than before, resulting in a decrease in the size of the guaranteed region of operation (see [20] for more details).

4 Real-Time Trajectory Generation and Differential Flatness

In this section we demonstrate how to use differential flatness to find fast numerical algorithms for solving optimal control problems. We consider the affine nonlinear control system

$$\dot{x} = f(x) + g(x)u, \quad (12)$$

where all vector fields and functions are smooth. For simplicity, we focus on the single input case, $u \in \mathbb{R}$. We wish to find a trajectory of equation (12) that minimizes the performance index (10), subject to a vector of initial, final, and trajectory constraints

$$\begin{aligned} lb_0 &\leq \psi_0(x(t_0), u(t_0)) \leq ub_0, \\ lb_f &\leq \psi_f(x(t_f), u(t_f)) \leq ub_f, \\ lb_t &\leq S(x, u) \leq ub_t, \end{aligned} \quad (13)$$

respectively. For conciseness, we will refer to this optimal control problem as

$$\min_{(x,u)} J(x, u) \quad \text{subject to} \quad \begin{cases} \dot{x} = f(x) + g(x)u, \\ lb \leq c(x, u) \leq ub. \end{cases} \quad (14)$$

4.1 Numerical Solution Using Collocation

A numerical approach to solving this optimal control problem is to use the direct collocation method outlined in Hargraves and Paris [12]. The idea behind this approach is to transform the optimal control problem into a nonlinear programming problem. This is accomplished by discretizing time into a grid of $N - 1$ intervals

$$t_0 = t_1 < t_2 < \dots < t_N = t_f \quad (15)$$

and approximating the state x and the control input u as piecewise polynomials \hat{x} and \hat{u} , respectively. Typically a cubic polynomial is chosen for the states and a linear polynomial for the control on each interval. Collocation is then used at the midpoint of each interval to satisfy equation (12). Let $\hat{x}(x(t_1), \dots, x(t_N))$ and $\hat{u}(u(t_1), \dots, u(t_N))$ denote the approximations to x and u , respectively, depending on $(x(t_1), \dots, x(t_N)) \in \mathbb{R}^{nN}$ and $(u(t_1), \dots, u(t_N)) \in \mathbb{R}^N$ corresponding to the value of x and u at the grid points. Then one solves the following finite dimension approximation of the original control problem (14):

$$\min_{y \in \mathbb{R}^M} F(y) = J(\hat{x}(y), \hat{u}(y)) \quad \text{subject to} \quad \begin{cases} \dot{\hat{x}} - f(\hat{x}(y)) + g(\hat{x}(y))\hat{u}(y) = 0, \\ lb \leq c(\hat{x}(y), \hat{u}(y)) \leq ub, \\ \forall t = \frac{t_j + t_{j+1}}{2} \quad j = 1, \dots, N - 1 \end{cases} \quad (16)$$

where $y = (x(t_1), u(t_1), \dots, x(t_N), u(t_N))$, and $M = \dim y = (n + 1)N$.

Seywald [28] suggested an improvement to the previous method (see also [2] page 362). Following this work, one first solves a subset of system dynamics in (14) for the the control in terms of combinations of the state and its time derivative. Then one substitutes for the control in the remaining system dynamics and constraints. Next all the time derivatives \dot{x}_i are approximated by the finite difference approximations

$$\dot{\hat{x}}(t_i) = \frac{x(t_{i+1}) - x(t_i)}{t_{i+1} - t_i}$$

to get

$$\left. \begin{aligned} p(\dot{x}(t_i), x(t_i)) &= 0 \\ q(\dot{x}(t_i), x(t_i)) &\leq 0 \end{aligned} \right\} \quad i = 0, \dots, N-1.$$

The optimal control problem is turned into

$$\min_{y \in \mathbb{R}^M} F(y) \quad \text{subject to} \quad \begin{cases} p(\dot{x}(t_i), x(t_i)) = 0 \\ q(\dot{x}(t_i), x(t_i)) \leq 0 \end{cases} \quad (17)$$

where $y = (x(t_1), \dots, x(t_N))$, and $M = \dim y = nN$. As with the Hargraves and Paris method, this parameterization of the optimal control problem (14) can be solved using nonlinear programming.

The dimensionality of this discretized problem is lower than the dimensionality of the Hargraves and Paris method, where both the states and the input are the unknowns. This induces substantial improvement in numerical implementation.

4.2 Differential Flatness Based Approach

The results of Seywald give a constrained optimization problem in which we wish to minimize a cost functional subject to $n-1$ equality constraints, corresponding to the system dynamics, at each time instant. In fact, it is usually possible to reduce the dimension of the problem further. Given an output, it is generally possible to parameterize the control and a part of the state in terms of this output and its time derivatives. In contrast to the previous approach, one must use more than one derivative of this output for this purpose.

When the whole state and the input can be parameterized with one output, the system is differentially flat, as described in Section 2. When the parameterization is only partial, the dimension of the subspace spanned by the output and its derivatives is given by r the *relative degree* of this output [14]. In this case, it is possible to write the system dynamics as

$$\begin{aligned} x &= \alpha(z, \dot{z}, \dots, z^{(q)}) \\ u &= \beta(z, \dot{z}, \dots, z^{(q)}) \\ \Phi(z, \dot{z}, \dots, z^{n-r}) &= 0 \end{aligned} \quad (18)$$

where $z \in \mathbb{R}^p$, $p > m$ represents a set of outputs that parameterize the trajectory and $\Phi : \mathbb{R}^n \times \mathbb{R}^m$ represents $n-r$ remaining differential constraints on the output. In the case that the system is flat, $r = n$ and we eliminate these differential constraints.

Unlike the approach of Seywald, it is not realistic to use finite difference approximations as soon as $r > 2$. In this context, it is convenient to represent z using B-splines. B-splines are chosen as basis functions because of their ease of enforcing continuity across knot points and ease of computing their derivatives. A pictorial representation of such an approximation is given in Figure 3. Doing so we get

$$z_j = \sum_{i=1}^{p_j} B_{i,k_j}(t) C_i^j, \quad p_j = l_j(k_j - m_j) + m_j$$

where $B_{i,k_j}(t)$ is the B-spline basis function defined in [6] for the output z_j with order k_j , C_i^j are the coefficients of the B-spline, l_j is the number of knot intervals, and m_j is number of smoothness conditions at the knots. The set $(z_1, z_2, \dots, z_{n-r})$ is thus represented by $M = \sum_{j \in \{1, r+1, \dots, n\}} p_j$ coefficients.

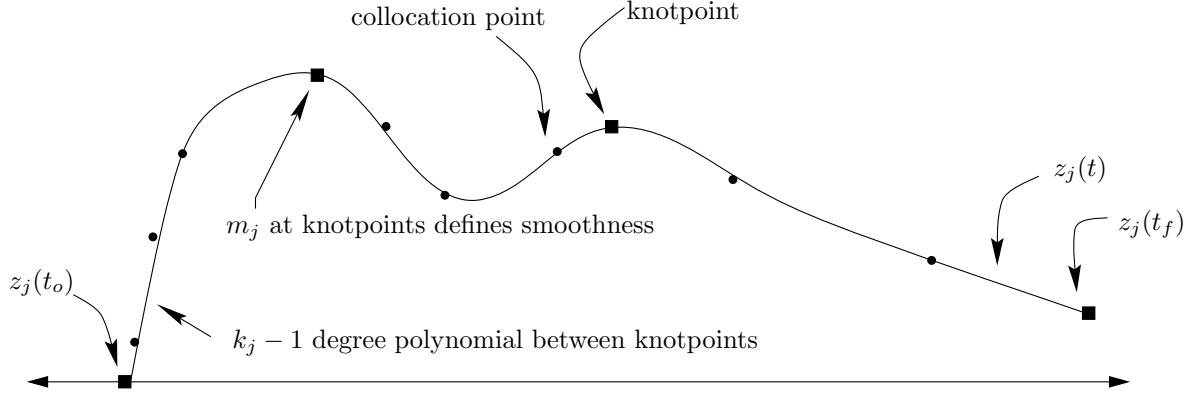


Figure 3: Spline representation of a variable.

In general, w collocation points are chosen uniformly over the time interval $[t_o, t_f]$ (though optimal knots placements or Gaussian points may also be considered). Both dynamics and constraints will be enforced at the collocation points. The problem can be stated as the following nonlinear programming form:

$$\min_{y \in \mathbb{R}^M} F(y) \quad \text{subject to} \quad \begin{cases} \Phi(z(y), \dot{z}(y), \dots, z^{(n-r)}(y)) = 0 \\ lb \leq c(y) \leq ub \end{cases} \quad (19)$$

where

$$y = (C_1^1, \dots, C_{p_1}^1, C_1^{r+1}, \dots, C_{p_{r+1}}^{r+1}, \dots, C_1^n, \dots, C_{p_n}^n).$$

The coefficients of the B-spline basis functions can be found using nonlinear programming.

A software package called Nonlinear Trajectory Generation (NTG) has been written to solve optimal control problems in the manner described above (see [23] for details). The sequential quadratic programming package NPSOL by [11] is used as the nonlinear programming solver in NTG. When specifying a problem to NTG, the user is required to state the problem in terms of some choice of outputs and its derivatives. The user is also required to specify the regularity of the variables, the placement of the knot points, the order and regularity of the B-splines, and the collocation points for each output.

5 Implementation on the Caltech Ducted Fan

To demonstrate the use of the techniques described in the previous section, we present an implementation of optimization-based control on the Caltech Ducted Fan, a real-time, flight control experiment that mimics the longitudinal dynamics of an aircraft. The experiment is shown in Figure 4.

5.1 Description of the Caltech Ducted Fan Experiment

The Caltech ducted fan is an experimental testbed designed for research and development of nonlinear flight guidance and control techniques for Uninhabited Combat Aerial Vehicles (UCAVs). The fan is a scaled model of the longitudinal axis of a flight vehicle and flight test results validate that the dynamics replicate qualities of actual flight vehicles [22].

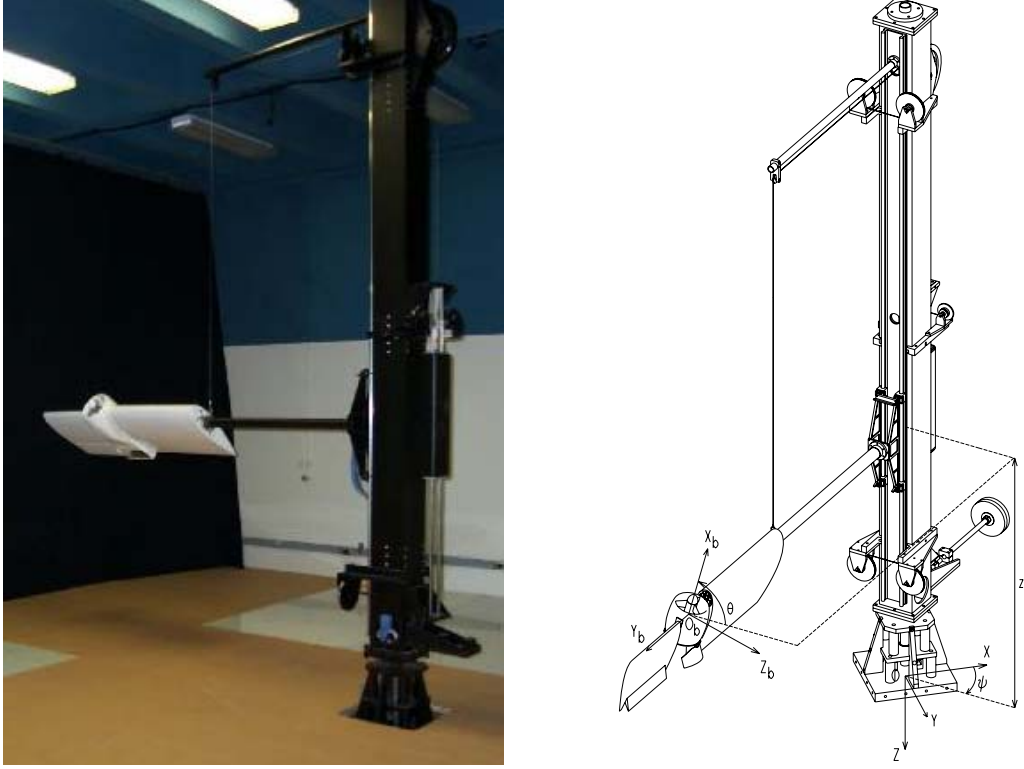


Figure 4: Caltech Ducted Fan.

The ducted fan has three degrees of freedom: the boom holding the ducted fan is allowed to operate on a cylinder, 2 m high and 4.7 m in diameter, permitting horizontal and vertical displacements. Also, the wing/fan assembly at the end of the boom is allowed to rotate about its center of mass. Optical encoders mounted on the ducted fan, gearing wheel, and the base of the stand measure the three degrees of freedom. The fan is controlled by commanding a current to the electric motor for fan thrust and by commanding RC servos to control the thrust vectoring mechanism.

The sensors are read and the commands sent by a dSPACE multi-processor system, comprised of a D/A card, a digital I/O card, two Texas Instruments C40 signal processors, two Compaq Alpha processors, and a ISA bus to interface with a PC. The dSPACE system provides a real-time interface to the 4 processors and I/O card to the hardware. The NTG software resides on both of the Alpha processors, each capable of running real-time optimization.

The ducted fan is modeled in terms of the position and orientation of the fan, and their velocities. Letting x represent the horizontal translation, z the vertical translation and θ the rotation about the boom axis, the equations of motion are given by

$$\begin{aligned}
 m\ddot{x} + F_{X_a} - F_{X_b} \cos \theta - F_{Z_b} \sin \theta &= 0 \\
 m\ddot{z} + F_{Z_a} + F_{X_b} \sin \theta - F_{Z_b} \cos \theta &= mg_{\text{eff}} \\
 J\ddot{\theta} - M_a + \frac{1}{r_s} I_p \Omega \dot{x} \cos \theta - F_{Z_b} r_f &= 0,
 \end{aligned} \tag{20}$$

where $F_{X_a} = D \cos \gamma + L \sin \gamma$ and $F_{Z_a} = -D \sin \gamma + L \cos \gamma$ are the aerodynamic forces and F_{X_b} and F_{Z_b} are thrust vectoring body forces in terms of the lift (L), drag (D), and flight path angle (γ).

I_p and Ω are the moment of inertia and angular velocity of the ducted fan propeller, respectively. J is the moment of ducted fan and r_f is the distance from center of mass along the X_b axis to the effective application point of the thrust vectoring force. The angle of attack α can be derived from the pitch angle θ and the flight path angle γ by

$$\alpha = \theta - \gamma.$$

The flight path angle can be derived from the spatial velocities by

$$\gamma = \arctan \frac{-\dot{z}}{\dot{x}}.$$

The lift (L), drag (D), and moment (M) are given by

$$L = qSC_L(\alpha) \quad D = qSC_D(\alpha) \quad M = \bar{c}SC_M(\alpha),$$

respectively. The dynamic pressure is given by $q = \frac{1}{2}\rho V^2$. The norm of the velocity is denoted by V , S the surface area of the wings, and ρ is the atmospheric density. The coefficients of lift ($C_L(\alpha)$), drag ($C_D(\alpha)$) and the moment coefficient ($C_M(\alpha)$) are determined from a combination of wind tunnel and flight testing and are described in more detail in [22], along with the values of the other parameters.

5.2 Real-Time Trajectory Generation

In this section we describe the implementation of a two degree of freedom controller using NTG to generate minimum time trajectories in real time. We first give a description of the controllers and observers necessary for stabilization about the reference trajectory, and discuss the NTG setup used for the forward flight mode. Finally, we provide example trajectories using NTG for the forward flight mode on the Caltech Ducted Fan experiment.

Stabilization Around Reference Trajectory

Although the reference trajectory is a feasible trajectory of the model, it is necessary to use a feedback controller to counteract model uncertainty. There are two primary sources of uncertainty in our model: aerodynamics and friction. Elements such as the ducted fan flying through its own wake, ground effects, and thrust not modeled as a function of velocity and angle of attack contribute to the aerodynamic uncertainty. The friction in the vertical direction is also not considered in the model. The prismatic joint has an unbalanced load creating an effective moment on the bearings. The vertical frictional force of the ducted fan stand varies with the vertical acceleration of the ducted fan as well as the forward velocity. Actuation models are not used when generating the reference trajectory, resulting in another source of uncertainty.

The separation principle was kept in mind when designing the observer and controller. Since only the position of the fan is measured, we must estimate the velocities. The observer that works best to date is an extended Kalman filter. The optimal gain matrix is gain scheduled on the (estimated) forward velocity. The Kalman filter outperformed other methods that computed the derivative using only the position data and a filter.

The stabilizing LQR controllers were gain scheduled on pitch angle, θ , and the forward velocity, \dot{x} . The pitch angle was allowed to vary from $-\pi/2$ to $\pi/2$ and the velocity ranged from 0 to 6 m/s. The weights were chosen differently for the hover-to-hover and forward flight modes. For the forward flight mode, a smaller weight was placed on the horizontal (x) position of the fan

compared to the hover-to-hover mode. Furthermore, the z weight was scheduled as a function of forward velocity in the forward flight mode. There was no scheduling on the weights for hover-to-hover. The elements of the gain matrices for each of the controller and observer are linearly interpolated over 51 operating points.

Nonlinear Trajectory Generation Parameters

We solve a minimum time optimal control problem to generate a feasible trajectory for the system. The system is modeled using the nonlinear equations described above and computed the open loop forces and state trajectories for the nominal system. The three outputs $z_1 = x$, $z_2 = z$, and $z_3 = \theta$ are each parameterized with four (intervals), sixth order, C^4 (multiplicity), piecewise polynomials over the time interval scaled by the minimum time. The last output ($z_4 = T$), representing the time horizon to be minimized, is parameterized by a scalar. By choosing the outputs to be parameterized in this way, we are in effect controlling the frequency content of inputs. Since we are not including the actuators in the model, it would be undesirable to have inputs with a bandwidth higher than the actuators. There are a total of 37 variables in this optimization problem. The trajectory constraints are enforced at 21 equidistant breakpoints over the scaled time interval.

There are many considerations in the choice of the parameterization of the outputs. Clearly there is a trade between the parameters (variables, initial values of the variables, and breakpoints) and measures of performance (convergence, run-time, and conservative constraints). Extensive simulations were run to determine the right combination of parameters to meet the performance goals of our system.

Forward Flight

To obtain the forward flight test data, the operator commanded a desired forward velocity and vertical position with the joysticks. We set the trajectory update time, δ to 2 seconds. By rapidly changing the joysticks, NTG produces high angle of attack maneuvers. Figure 5(a) depicts the reference trajectories and the actual θ and \dot{x} over 60 sec. Figure 5(b) shows the commanded forces for the same time interval. The sequence of maneuvers corresponds to the ducted fan transitioning from near hover to forward flight, then following a command from a large forward velocity to a large negative velocity, and finally returning to hover.

Figure 6 is an illustration of the ducted fan altitude and x position for these maneuvers. The air-foil in the figure depicts the pitch angle (θ). It is apparent from this figure that the stabilizing controller is not tracking well in the z direction. This is due to the fact that unmodeled frictional effects are significant in the vertical direction. This could be corrected with an integrator in the stabilizing controller.

An analysis of the run times was performed for 30 trajectories; the average computation time was less than one second. Each of the 30 trajectories converged to an optimal solution and was approximately between 4 and 12 seconds in length. A random initial guess was used for the first NTG trajectory computation. Subsequent NTG computations used the previous solution as an initial guess. Much improvement can be made in determining a “good” initial guess. Improvement in the initial guess will improve not only convergence but also computation times.

5.3 Model Predictive Control

The results of the previous section demonstrate the ability to compute optimal trajectories in real time, although the computation time was not sufficiently fast for closing the loop around the

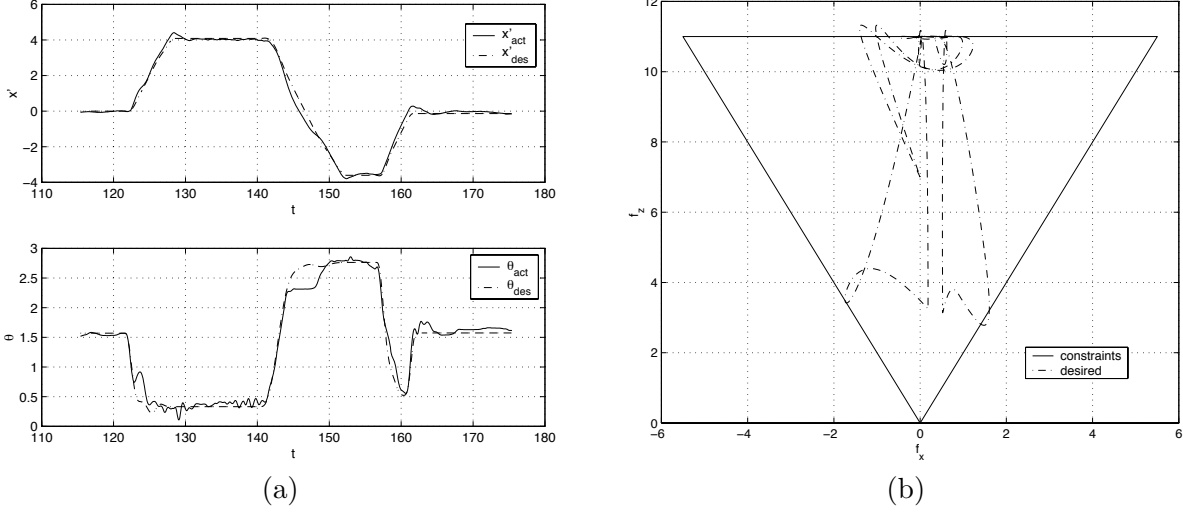


Figure 5: Forward Flight Test Case: (a) θ and \dot{x} desired and actual, (b) desired F_{X_b} and F_{Z_b} with bounds.

optimization. In this section, we make use of a shorter update time δ , a fixed horizon time T with a quadratic integral cost, and a CLF terminal cost to implement the receding horizon controller described in Section 3.

We have implemented the receding horizon controller on the ducted fan experiment where the control objective is to stabilize the hover equilibrium point. The quadratic cost is given by

$$\begin{aligned} q(x, u) &= \frac{1}{2} \hat{x}^T Q \hat{x} + \frac{1}{2} \hat{u}^T R \hat{u} \\ V(x) &= \gamma \hat{x}^T P \hat{x} \end{aligned} \quad (21)$$

where

$$\begin{aligned} \hat{x} &= x - x_{eq} = (x, z, \theta - \pi/2, \dot{x}, \dot{z}, \dot{\theta}) \\ \hat{u} &= u - u_{eq} = (F_{X_b} - mg, F_{Z_b}) \\ Q &= \text{diag}\{4, 15, 4, 1, 3, 0.3\} \\ R &= \text{diag}\{0.5, 0.5\}, \end{aligned}$$

$\gamma = 0.075$ and P is the unique stable solution to the algebraic Riccati equation corresponding to the linearized dynamics of equation (3) at hover and the weights Q and R . Note that if $\gamma = 1/2$, then $V(\cdot)$ is the CLF for the system corresponding to the LQR problem. Instead V is a relaxed (in magnitude) CLF, which achieved better performance in the experiment. In either case, V is valid as a CLF only in a neighborhood around hover since it is based on the linearized dynamics. We do not try to compute off-line a region of attraction for this CLF. Experimental tests omitting the terminal cost and/or the input constraints leads to instability. The results in this section show the success of this choice for V for stabilization. An inner-loop PD controller on $\theta, \dot{\theta}$ is implemented to stabilize to the receding horizon states $\theta_T^*, \dot{\theta}_T^*$. The θ dynamics are the fastest for this system and although most receding horizon controllers were found to be nominally stable without this inner-loop controller, small disturbances could lead to instability.

The optimal control problem is set-up in NTG code by parameterizing the three position states (x, z, θ) , each with 8 B-spline coefficients. Over the receding horizon time intervals, 11 and 16

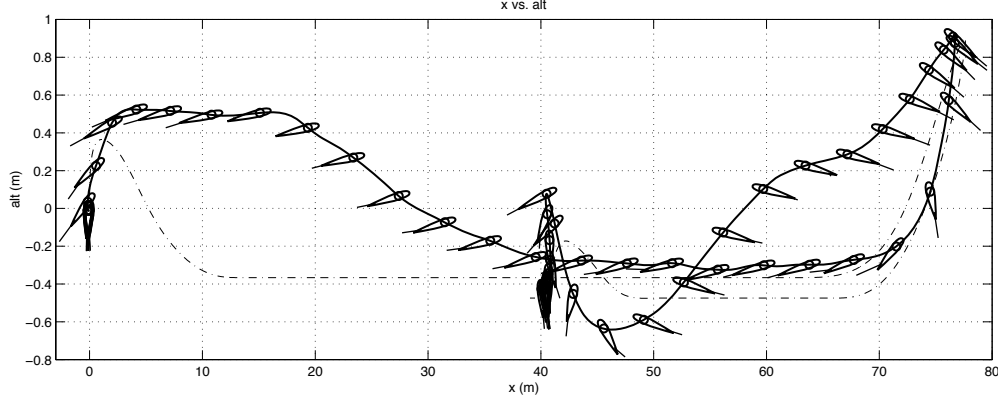


Figure 6: Forward Flight Test Case: Altitude and x position (actual (solid) and desired (dashed)). Airfoil represents actual pitch angle (θ) of the ducted fan.

breakpoints were used with horizon lengths of 1, 1.5, 2, 3, 4 and 6 seconds. Breakpoints specify the locations in time where the differential equations and any constraints must be satisfied, up to some tolerance. The value of $F_{X_b}^{\max}$ for the input constraints is made conservative to avoid prolonged input saturation on the real hardware. The logic for this is that if the inputs are saturated on the real hardware, no actuation is left for the inner-loop θ controller and the system can go unstable. The value used in the optimization is $F_{X_b}^{\max} = 9$ N.

Computation time is non-negligible and must be considered when implementing the optimal trajectories. The computation time varies with each optimization as the current state of the ducted fan changes. The following notational definitions will facilitate the description of how the timing is set-up:

i	Integer counter of MPC computations
t_i	Value of current time when MPC computation i started
$\delta_c(i)$	Computation time for computation i
$u_T^*(i)(t)$	Optimal output trajectory corresponding to computation i , with time interval $t \in [t_i, t_i + T]$

A natural choice for updating the optimal trajectories for stabilization is to do so as fast as possible. This is achieved here by constantly resolving the optimization. When computation i is done, computation $i + 1$ is immediately started, so $t_{i+1} = t_i + \delta_c(i)$. Figure 7 gives a graphical picture of the timing set-up as the optimal input trajectories $u_T^*(\cdot)$ are updated. As shown in the figure, any computation i for $u_T^*(i)(\cdot)$ occurs for $t \in [t_i, t_{i+1}]$ and the resulting trajectory is applied for $t \in [t_{i+1}, t_{i+2}]$. At $t = t_{i+1}$ computation $i + 1$ is started for trajectory $u_T^*(i+1)(\cdot)$, which is applied as soon as it is available ($t = t_{i+2}$). For the experimental runs detailed in the results, $\delta_c(i)$ is typically in the range of $[0.05, 0.25]$ seconds, meaning 4 to 20 optimal control computations per second. Each optimization i requires the current measured state of the ducted fan and the value of the previous optimal input trajectories $u_T^*(i-1)$ at time $t = t_i$. This corresponds to, respectively, 6 initial conditions for state vector x and 2 initial constraints on the input vector u . Figure 7 shows that the optimal trajectories are advanced by their computation time prior to application to the system. A dashed line corresponds to the initial portion of an optimal trajectory and is not applied since it is not available until that computation is complete. The figure also reveals the possible discontinuity between successive applied optimal input trajectories, with a larger discontinuity

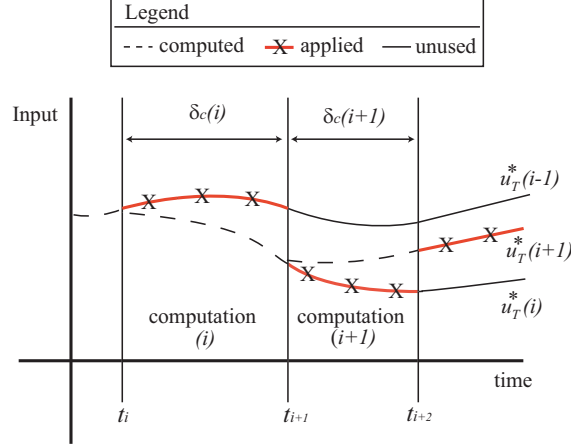


Figure 7: Receding horizon input trajectories

more likely for longer computation times. The initial input constraint is an effort to reduce such discontinuities, although some discontinuity is unavoidable by this method. Also note that the same discontinuity is present for the 6 open-loop optimal state trajectories generated, again with a likelihood for greater discontinuity for longer computation times. In this description, initialization is not an issue because we assume the receding horizon computations are already running prior to any test runs. This is true of the experimental runs detailed in the results.

The experimental results show the response of the fan with each controller to a 6 meter horizontal offset, which is effectively engaging a step-response to a change in the initial condition for x . The following details the effects of different receding horizon control parameterizations, namely as the horizon changes, and the responses with the different controllers to the induced offset.

The first comparison is between different receding horizon controllers, where time horizon is varied to be 1.5, 2.0, 3.0, 4.0 or 6.0 seconds. Each controller uses 16 breakpoints. Figure 8(a) shows a comparison of the average computation time as time proceeds. For each second after the offset was initiated, the data corresponds to the average run time over the previous second of computation. Note that these computation times are substantially smaller than those reported for real-time trajectory generation, due to the use of the CLF terminal cost versus the terminal constraints in the minimum-time, real-time trajectory generation experiments.

There is a clear trend toward shorter average computation times as the time horizon is made longer. There is also an initial transient increase in average computation time that is greater for shorter horizon times. In fact, the 6 second horizon controller exhibits a relatively constant average computation time. One explanation for this trend is that, for this particular test, a 6 second horizon is closer to what the system can actually do. After 1.5 seconds, the fan is still far from the desired hover position and the terminal cost CLF is large, likely far from its region of attraction. Figure 8(b) shows the measured x response for these different controllers, exhibiting a rise time of 8–9 seconds independent of the controller. So a horizon time closer to the rise time results in a more feasible optimization in this case.

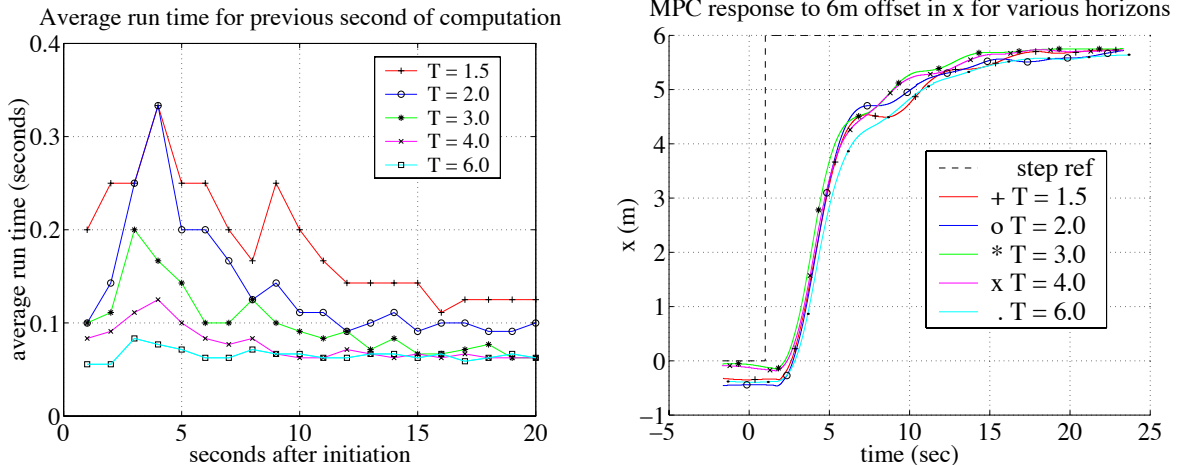


Figure 8: Receding horizon control: (a) moving one second average of computation time for MPC implementation with varying horizon time, (b) response of MPC controllers to 6 meter offset in x for different horizon lengths.

6 Summary and Conclusions

This chapter has given a survey of some basic concepts required to analyze and implement on-line control customization via optimization-based control. By making use of real-time trajectory generation algorithms that exploit geometric structure and implementing receding horizon control using control Lyapunov functions as terminal costs, we have been able to demonstrate closed-loop control on a flight control experiment. These results build on the rapid advances in computational capability over the past decade, combined with careful use of control theory, system structure, and numerical optimization. A key property of this approach is that it explicitly handles constraints in the input and state vectors, allowing complex nonlinear behavior over large operating regions.

The framework presented here is a first step toward a fundamental shift in the way that control laws are designed and implemented. By moving the control design into the system itself, it becomes possible to implement much more versatile controllers that respond to changes in the system dynamics, mission intent, and environmental constraints. Experimental results have validated this approach in the case of manually varied end points, a particularly simple version of change in mission.

Future control systems will continue to be more complex and more interconnected. An important element will be the networked nature of future control systems, where many individual agents are combined to allow cooperative control in dynamic, uncertain, and adversarial environments. While many traditional control paradigms will not operate well for these classes of systems, the optimization-based controllers presented here can be transitioned to systems with strongly nonlinear behavior, communications delays, and mixed continuous and discrete states. Thus, while traditional frequency domain techniques are likely to remain useful for isolated systems, design of controllers for large-scale, complex, networked systems will increasingly rely on techniques based on Lyapunov theory and (closed loop) optimal control.

However, there are still many gaps in the theory and practice of optimization-based control. Guaranteed robustness, the hallmark of modern control theory, is largely absent from our present formulation and will require substantial work in extending the theory. Existing approaches such as

differential games are not likely to work in an online environment, due to the extreme computational cost required to solve such problems. Furthermore, while the extension to hybrid systems with mixed continuous and discrete variables (in both states and time) is conceivable at the theoretical level, effective computational tools for mixed integer programs must be developed that exploit the system structure to achieve fast computation.

Finally, we note that existing optimization-based techniques are still primarily aimed at the lowest levels of control, despite their potential to apply more broadly. Higher level protocols for control and decision making must be developed that build on the strength of optimization-based control, but they are likely to require substantially new paradigms and approaches. At the same time, new methods in designing software systems that take into account external dynamics and environmental factors are also required.

Acknowledgments

The authors would like to thank Mario Sznaier and John Doyle for many helpful discussions on the results presented here. The support of the Software Enabled Control (SEC) program and the SEC team members is also gratefully acknowledged.

References

- [1] A. E. Bryson. *Dynamic optimization*. Addison Wesley, 1999.
- [2] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.
- [3] W. B. Dunbar, M. B. Milam, R. Franz, and R. M. Murray. Model predictive control of a thrust-vectoring flight control experiment. In *Proc. IFAC World Congress*, 2002. Submitted.
- [4] M. Fliess, J. Levine, P. Martin, and P. Rouchon. On differentially flat nonlinear systems. *Comptes Rendus des Séances de l'Académie des Sciences*, 315:619–624, 1992. Serie I.
- [5] R. Franz, M. B. Milam, and J. E. Hauser. Applied receding horizon control of the caltech ducted fan. In *Proc. American Control Conference*, 2002. Submitted.
- [6] P. E. Gill, W. Murray, M. A. Saunders, and M. Wright. *User's Guide for NPSOL 5.0: A Fortran Package for Nonlinear Programming*. Systems Optimization Laboratory, Stanford University, Stanford, CA 94305.
- [7] C. Hargraves and S. Paris. Direct trajectory optimization using nonlinear programming and collocation. *AIAA J. Guidance and Control*, 10:338–342, 1987.
- [8] J. Hauser and H. Osinga. On the geometry of optimal control: the inverted pendulum example. In *American Control Conference*, 2001.
- [9] A. Isidori. *Nonlinear Control Systems*. Springer-Verlag, 2nd edition, 1989.
- [10] A. Jadbabaie. *Nonlinear Receding Horizon Control: A Control Lyapunov Function Approach*. PhD thesis, California Institute of Technology, Control and Dynamical Systems, 2001.
- [11] A. Jadbabaie, J. Yu, and J. Hauser. Unconstrained receding horizon control of nonlinear systems. *IEEE Transactions on Automatic Control*, 46, May 2001.

- [12] M. Krstić, I. Kanellakopoulos, and P. Kokotović. *Nonlinear and Adaptive Control Design*. Wiley, 1995.
- [13] E. B. Lee and L. Markus. *Foundations of Optimal Control Theory*. Wiley, New York, 1967.
- [14] P. Martin, S. Devasia, and B. Paden. A different look at output tracking—Control of a VTOL aircraft. *Automatica*, 32(1):101–107, 1994.
- [15] D. Q. Mayne, J. B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.
- [16] M. B. Milam, R. Franz, and R. M. Murray. Real-time constrained trajectory generation applied to a flight control experiment. In *Proc. IFAC World Congress*, 2002. Submitted.
- [17] M. B. Milam and R. M. Murray. A testbed for nonlinear flight control techniques: The Caltech ducted fan. In *Proc. IEEE International Conference on Control and Applications*, 1999.
- [18] M. B. Milam, K. Mushambi, and R. M. Murray. A computational approach to real-time trajectory generation for constrained mechanical systems. In *Proc. IEEE Control and Decision Conference*, 2000.
- [19] R. M. Murray. Nonlinear control of mechanical systems: A Lagrangian perspective. *Annual Reviews in Control*, 21:31–45, 1997.
- [20] N. Petit, M. B. Milam, and R. M. Murray. Inversion based trajectory optimization. In *IFAC Symposium on Nonlinear Control Systems Design (NOLCOS)*, 2001.
- [21] J. A. Primbs, V. Nevistić, and J. C. Doyle. A receding horizon generalization of pointwise min-norm controllers. *IEEE Transactions on Automatic Control*, 45:898–909, June 2000.
- [22] S.J. Qin and T.A. Badgwell. An overview of industrial model predictive control technology. In J.C. Kantor, C.E. Garcia, and B. Carnahan, editors, *Fifth International Conference on Chemical Process Control*, pages 232–256, 1997.
- [23] H. Seywald. Trajectory optimization based on differential inclusion. *J. Guidance, Control and Dynamics*, 17(3):480–487, 1994.
- [24] L. Singh and J. Fuller. Trajectory generation for a uav in urban terrain, using nonlinear mpc. In *Proceedings of the American Control Conference*, 2001.
- [25] E. D. Sontag. A Lyapunov-like characterization of asymptotic controllability. *SIAM Journal of Control and Optimization*, 21:462–471, 1983.
- [26] M. J. van Nieuwstadt and R. M. Murray. Rapid hover to forward flight transitions for a thrust vectored aircraft. *Journal of Guidance, Control, and Dynamics*, 21(1):93–100, 1998.
- [27] M. J. van Nieuwstadt and R. M. Murray. Real time trajectory generation for differentially flat systems. *International Journal of Robust and Nonlinear Control*, 8(11):995–1020, 1998.

Distributed Computation for Cooperative Control

Eric Klavins

Electrical Engineering
University of Washington
Seattle, WA
 klavins@u.washington.edu

Richard M. Murray

Control and Dynamical Systems
California Institute of Technology
Pasadena, CA
 murray@cds.caltech.edu

Abstract

A cooperative control system consists of multiple, autonomous components interacting to control their environment. Examples include air traffic control systems, automated factories, robot soccer teams and sensor/actuator networks. Designing such systems requires a combination of tools from control theory and distributed systems. In this article, we review some of these tools and then focus on the Computation and Control Language, CCL, which we have developed as a modeling tool and a programming language for cooperative control systems.

1 Introduction

A cooperative control system consists of multiple, autonomous components interacting to control their environment. Examples include air traffic control systems, automated factories, robot soccer teams and sensor/actuator networks. In each of these systems, a component reacts to its environment and to messages received from neighboring components. Thus, a cooperative control system is at once a controlled physical system and a distributed computer. Designing cooperative control systems, therefore, requires a combination of tools from control theory and distributed systems.

A motivating example for our work is the *RoboFlag* game [3], a successor to the *RoboCup* robotic soccer tournament dedicated to the goal of building a robotic soccer team by 2050. RoboFlag is a version of “capture the flag”, using the setup illustrated in Figure 1. Each team (red and blue) has a home zone, a defensive zone containing a flag and between eight and ten robots. The game can be played either with autonomous controllers or with one or two humans in the loop giving high-level directives to their teams. The goal of the red team, say, is to capture the blue team’s flag and return it to the red home zone, meanwhile defending its own flag. If a red robot is “tagged” or touched by a blue robot while on the blue side of the field, it must return to its home zone for a “time out”. The blue robots have a symmetric goal.

In addition to having to control their own motions, the robots have limited sensing capabilities and are organized as a distributed computational system, requiring that information be communicated between robots and across limited bandwidth links. Each robot must, therefore, contain a program that allows it to control its motion, react to events near it and participate in group strategies. Designing such programs so that they are correct, robust and fault tolerant is the goal of cooperative control.

Control vs. Distributed Computation Two very different worlds collide in cooperative control problems such as the RoboFlag game. On one hand, things like robots are electro-mechanical objects

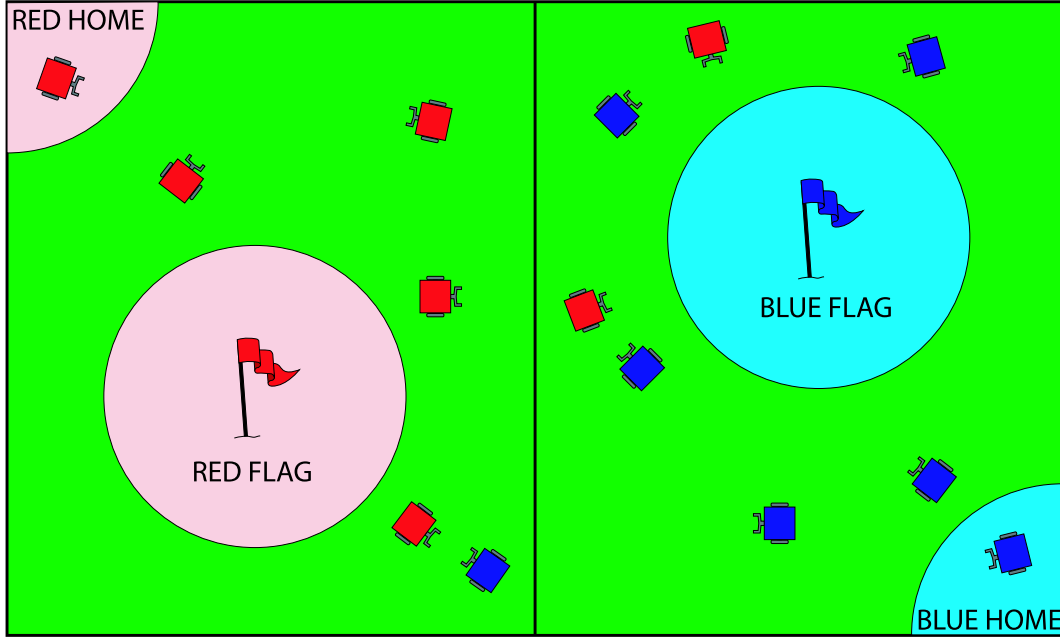


Figure 1: The *RoboFlag* game. Two teams of robots, red and blue, must defend their flags while attempting to capture the other team’s flag.

whose interactions with the physical world we would like to manage. We usually speak of control with respect to a dynamical model of the world, such as a set of differential equations with inputs and outputs. The control problem is that of “closing the loop”, that is, defining input rules as functions of the output values to produce a desired behavior. As control engineers, we worry about the stability, robustness and performance of the systems we design. On the other hand, a *group* of robots is by all rights a distributed computational system, each robot having its own processor and (presumably) some method of communicating with the processors on other robots. Presently, there are no universally agreed upon models for distributed systems: We might use *I/O automata*, *process algebras* or *guarded command languages* to describe how messages are passed between robots or how instructions on different processors may be interleaved. As distributed systems engineers, we worry about protocol design, deadlock avoidance and communication complexity.

The difficulty is that we cannot temporarily ignore one of these worlds while concentrating on design problems in the other. As cooperative control engineers, we must be concerned with communication protocols because they introduce delays, which are notorious for degrading performance and causing instabilities. We must mind the communication complexity of the system: A truly decentralized control algorithm will require only a few messages to be passed from robot to robot and in particular will not demand that each robot know the state of every other robot in order to act. Unfortunately, most tried and true control techniques are blind to these problems. As distributed systems engineers, we must design protocols that respect the dynamics of the environment: For example, a protocol intended to reach a consensus among a formation of aircraft about how to respond to a threat must finish before the momentum of the aircraft carries them inescapably close to the threat. In contrast, momentum and acceleration are often not a concern in traditional distributed systems wherein, for example, a bank customer can simply wait for a distributed online transaction to complete (his momentum being of no concern to the bank).

At the heart of the difference between control engineering and distributed system engineering lies the role of the environment in the design process. In control, the unpredictable, messy and incompletely understood environment is tightly coupled with a control process designed to reject disturbances from the environment to achieve a certain desired operating condition. For example, the autopilot on an aircraft will attempt to maintain altitude, heading and speed despite wind gusts and turbulence. The beauty of feedback control is that, to a large extent, it is robust to the differences in the mathematical model of how the environment affects the system and the actual effect of the environment on the system.

In contrast, in computational systems and distributed systems in particular, there often is no explicit environment whatsoever, and the notion of robustness to modeling errors is not even an issue. Such systems consist of nothing but the internal states (memories, file systems) of the processes involved. The task for the distributed systems engineer is to manipulate this information and keep it consistent among the various processes. When the issue of robustness does arise, it is with respect to whether the system can continue to function in the event that one of the processors fails.

When we design multi-vehicle systems, sensor-actuator networks or automated factories we must merge these two ways of looking at the problem. A dynamical model of the response of the system to its environment is mandatory, and so is an understanding of how information flows from process to process. We must ask questions about the stability of motions in the environment and the stability of information in the network. We must ensure that the system is robust to disturbances both physical, such as resulting from a wind gust, and logical, such as resulting from a hard reboot of a processor.

Synopsis In this article we describe at a high level some of the methods that are used to bridge these two ways of modeling and designing systems. For the sake of brevity, our review is incomplete and biased toward our own work on the *Computation and Control Language* or *CCL*, which we have begun to use for modeling control systems, especially distributed ones. To illustrate some of the concepts involved, we present a fairly complete example of a multi-robot task, inspired by the RoboFlag scenario, and show what kinds of questions we can answer about the model. Finally, we discuss one of the features of CCL, which is that it can be used as a programming language as well as a modeling tool so that the models we write down in CCL can be directly simulated or executed on hardware.

2 Models

The first step toward determining that a cooperative control system has a given property is to *write down* a description, or model, of what the system actually is to some appropriate level of detail. A control engineer might supply you with a set of differential equations that describe the closed loop (system + control) dynamics of the system. Unfortunately, once the control rules have been implemented in a distributed fashion, a simple differential equations description of the system fails to capture many important qualities, as we noted in the introduction. Thus, this description must be combined in some way with a description of the distributed system that implements the control law and that accounts for the effects of “spreading” the control law out among multiple processors.

In this section we review several formalisms for writing down (or modeling) what computation and control systems system are and what distributed systems are, starting with Hybrid Automata, then I/O Automata, temporal logic finally and UNITY. Each of these formalisms have qualities

we need for modeling Multi-Vehicle Systems, but none is entirely adequate for our purposes. Our main goal in this section, besides review, is to shed light on several important issues that led us to our present use of the *Computation and Control Language* (CCL), which we describe in the next sections.

Hybrid Automata A popular way to write down a model of system that has both continuous dynamics and discrete “modes” of control is as a *Hybrid Automaton* or *HA* [1]. HAs come in many flavors and we summarize their commonalities here. A simple finite automaton consists of a finite set of states and a set of transitions between states. For example, the level of water in a leaky water tank may be increasing (*state one*) or decreasing (*state two*). Transitions between these two states might correspond to opening (*transition on*) or closing (*transition off*) an input valve on the tank. An HA extends the idea of a simple finite automaton with *continuous* variables that usually denote physical quantities (such as the exact level of water in the tank). An HA must say how its continuous variables change while any given state is active using a *differential inclusions* of the form

$$0 < \frac{d}{dt}h < 0.1$$

which might mean that the level of water h in a tank is increasing at a rate between 0 m/s and 0.1 m/s.

An HA assigns *guards* and *rules* to each transition. A *guard* is a predicate on the continuous state, such as $h > 5$ (read “the value of h is 5”). If the guard on a transition from state one to state two becomes true, then the discrete state changes from one to two. A *rule* is an assignment, such as $t := 0$ which might denote the resetting of a timer variable. When a transition is taken, any rules associated with it are executed.

An important aspect of HAs is that they can be *composed* to make larger models. Roughly, the composition of two HAs H_1 and H_2 is another HA denoted $H_1 || H_2$ whose state set is (more or less) the cross product of the state sets of its constituents. Any transitions from H_1 and H_2 that have the same label must synchronize. For example, a water tank controller might issue *on* and *off* commands that would be synchronized with (i.e. technically they are the same transition as) the *on* and *off* transitions in the water tank model. This form of composition is acceptable for small systems. However, it is awkward for the sorts of systems found in cooperative control. For example, suppose each robot in a multi-robot system is modeled by an automaton R_i with r states. A set of n robots is modeled by $R_1 || \dots || R_n$ which has r^n states. Furthermore, any transitions with the same label must be synchronized in the composition, which would seem to suggest that the robots are not entirely independent in this model.

I/O Automata A very successful tool for modeling distributed systems is the *I/O automaton* (IOA) model [9]. In it, an individual component is modeled as an automaton as above, except with possibly an infinite set of states. Transitions, called *actions* in IOA theory, are labeled as either *input*, *output* or *internal*. The composition of multiple components is much different than the cross product composition discussed above, however. If an IOA with output action a is composed with other IOAs, then the other IOAs must label a as an input action. An execution of an IOA consists of a sequence of actions taken by the components one at a time. A component may execute an internal action, in which case only its local state changes. A component may also execute an output action, say a , that causes all other components with (input) actions labeled by a to synchronously execute their local copies of action a , thereby changing their states. The one restriction, called a *fairness constraint*, is that each component must be allowed to take a non-input action infinitely often in

any execution. The result is an *interleaving* of actions taken by each component, with occasional partial synchronization of some of the components. In the example above, the composition of n robots in this model would have an n -dimensional vector describing its state (still living in an r^n sized space, of course, but somehow more parsimonious). Furthermore, the interleaving execution model more naturally reflects the possible ways that individual components may execute in parallel. In particular, a property P of an IOA is said to hold if and only if it holds for *all possible* fairly interleaved executions and is, therefore, robust in some sense to how the actions of the components are scheduled. IOAs have been used extensively to model distributed algorithms [9] and have proved quite amenable to analysis.

The IOA model has been extended to handle systems with continuous state variables that change according to differential equations. The result is the very comprehensive, if somewhat sophisticated, Hybrid I/O Automata (HIOA) Model [10]. In the HIOA model, continuous time variables follow trajectories according to the equations corresponding to the state of the HIOA. The trajectories are punctuated by actions taken by the various components. Because the continuous time variables of each component evolve in parallel, however, this can lead to very complex overall trajectories that are difficult to reason about.

Temporal Logic An important tool used to describe distributed systems is *temporal logic*. In temporal logic, we reason about the possible behaviors of a system (such as arising from an automaton model or a program written in Java, for example). Behaviors are defined to be sequences of states of a system. A state s , essentially a “snapshot” of a system, might assign the value of a variable x to 7 and the value of y to *true*. A behavior describes how the values of x and y change. It is important to note that there is no notion of continuous time *per se* in temporal logic, only the notion that a given state comes before or after some other state in a behavior.

Formulas in temporal logic are of the form “always P ” (written $\Box P$) or “eventually Q ” (or $\Diamond Q$), where P and Q are predicates on states. For example, if σ is the behavior

$$x := 1, x := 2, x := 3, \dots,$$

assigning x to k in σ_k , then the statement $\Box x > 0$ is true of σ while the statement $\Diamond x < 0$ is false of σ . Temporal logic also defines the notion of an *action* as a relation between states. We usually write, for example, $x' = x + 1$ to denote relations between states and say that s is related to t by the action $x' = x + 1$ if the value of x in state t is equal to the value of x in state s plus one. For example, $\Box x' = x + 1$ is true of σ as defined above. Temporal logic can also be used to reason about real-time and hybrid systems with the careful use of *time* variables and yet without any further formal machinery [7].

A temporal logic formula F *specifies* a set of allowable behaviors: those behaviors for which F is true. Thus, we usually call F a specification instead of a formula. If F and G are specifications then $F \Rightarrow G$ is true if all the behaviors of F are also behaviors of G . We say that F meets the specification G . An implementation, or program, is then essentially a temporal logic formula that admits only one behavior for any initial state. Furthermore, specifications may be composed by simple conjunction. In our multi-robot example, if R_1, \dots, R_n are specifications of individual robot behaviors, then $R_1 \wedge \dots \wedge R_n$ is their composition.

A complex temporal logic formula usually consists of two parts: a *safety* specification and a *fairness* constraint. The safety specification is used to state what actions the components of the system are allowed to take to yield new states. The fairness constraint states when a component may take an action — infinitely often, for example. In a super simple multi-robot system, a robot

i might be described by the formula

$$\Box(x'_i = x_i + 1 \vee x'_i = x_i) \wedge \Box\Diamond(x'_i \neq x_i)$$

which states that the robot may move forward or stay still (safety) and that eventually it must move (fairness). Fairness constraints tend to get fairly complex, especially when real time is considered, and are the main source of complexity in temporal logic specifications.

In CCL, which we describe below, temporal logic is the tool we use to state the properties of the programs we write. A particularly important property is the stability of a predicate. For example, the formula

$$\Diamond\Box|x_i| < \varepsilon$$

states that x_i (a robot's position, say) is eventually always less than ε in magnitude.

UNITY The *non-duality* of programs and specifications (mentioned above) is heralded as the beauty of temporal logic and has been used with great success to reason about concurrent systems. An especially useful result of non-duality is the ease with which specifications may be automatically verified using a combination of model checking and automated theorem proving. However, a complication of the safety-fairness way of writing a specification is that it results in formulas that do not look very much like programs. In fact, duality may make life simpler. This is the approach taken by the *UNITY* formalism for parallel program design [2].

In *UNITY*, specifications S are written as a set of (possibly guarded) variable assignments. To arrive at a behavior, we simply start with some initial state, and then non-deterministically pick assignments one at a time from the set and apply them to the state to get a sequence of states. The only requirement is that each assignment is applied infinitely often in any behavior. *UNITY* is thus a kind of theoretical programming language that runs on an odd sort of non-deterministic machine in which a particular fairness constraint is built-in. As with CCL (described below), temporal logic turns out to be the most convenient way to reason about specifications. The general goal is to determine when a formula F is true of every behavior allowed by a specification S .

In controls, we often imagine that the components of a system are all executing their instructions at more or less the same frequency. So the fairness constraint adopted by *UNITY* (and IOAs) that merely states “each process gets to execute *eventually*” is somewhat too relaxed. Furthermore, writing more complicated fairness constraints in temporal logic, such as will be discussed below, can be rather cumbersome. This was a main motivation for our developing CCL, which we describe next.

3 CCL

The *Computation and Control Language*, or CCL, is a modeling language similar in appearance to *UNITY*, but interpreted differently. The basic unit of a CCL program is the *guarded command* (or simply command) which we describe by example. Formal definitions can be found elsewhere [6]. An example of a guarded command is:

$$t > 10 : x' \geq x + 1 \wedge t' = 0.$$

The part before the colon is called the guard and the part after it is called the rule. We interpret it as follows: If this command is executed in a state where the variable t is greater than 10, then a new state will result in which the new value of x is greater than or equal to its old value plus 1,

and the new value of t is 0. All other variables (those not occurring *primed*) remain the same. If the command is executed in a state in which t is not greater than 10, then the new state is defined to be exactly the same as the old state. The execution of a command is called a *step*. Note that guarded commands can be non-deterministic, as is the one above since it does not specify the exact new value for x , only that it should increase by at least 1.

A complete CCL program $P = (I, C)$ consists of two parts: An initial predicate I that says what the initial values of the variables involved are allowed to be; and a set C of guarded commands. Here is an example program:

Program P	
Initial	$x_1 = x_2 = 0$
Clauses	$true : x'_1 = x_1 + \delta$
	$true : x'_2 = x_2 + \delta$

which, say, describes how the positions of two robots change. It says that initially, the robots are both at position zero and that they may move forward by δ meters upon taking a step.

CCL program composition is very straightforward. If $P_1 = (I_1, C_1)$ and $P_2 = (I_2, C_2)$, then their composition is simply $P_1 \circ P_2 = (I_1 \wedge I_2, C_1 \cup C_2)$. That is, to obtain the composition of two programs, conjoin their initial clauses and union their command sets.

A CCL program can be interpreted in various ways, depending on how its commands are scheduled for execution (or, equivalently, how we define fairness for the system). The most simple schedule is: starting with a state consistent with the initial predicate, the commands are executed in the order they were written down, over and over again. In this case, we could get something like the following execution for program P :

x_1	0	δ	δ	2δ	2δ	3δ	...
x_2	0	0	δ	δ	2δ	2δ	...

A more reasonable scheme, one that accounts for the robots not executing at the same speeds is called the *EPOCH* semantics:

EPOCH: All clauses in C must be executed before any of them can be executed again¹.

A subsequence where each clause has been executed exactly once (in any order) is called an *epoch*.

The *EPOCH* semantics allow for clauses to be executed in any order, as long as they are all “used up” before any get used again. A looser scheme is called partial synchronization, or *SYNCH*(τ) semantics, where τ is a positive integer:

SYNCH(τ): For any interval of a behavior and for any two commands, the difference between the number of times each command is executed during the interval must be less than or equal to τ .

Of course, in the limit as τ approaches infinity we obtain the familiar UNITY fairness constraint: that each clause must simply be executed infinitely often. Figure 2 illustrates these different interpretations with respect to the two-robot example.

As in UNITY, we express properties of CCL programs as temporal logic formulas and define $P \models_S F$, read P models F with semantics S , if F is true of all behaviors allowed by program P under the interpretation S . An instructive result is the following.

¹In all interpretations of CCL, a step may also execute no command at all, thereby leaving the state the same. This is called a *stutter step* and is useful for technical reasons beyond the scope of this article. See [8] for a discussion of stutter steps.

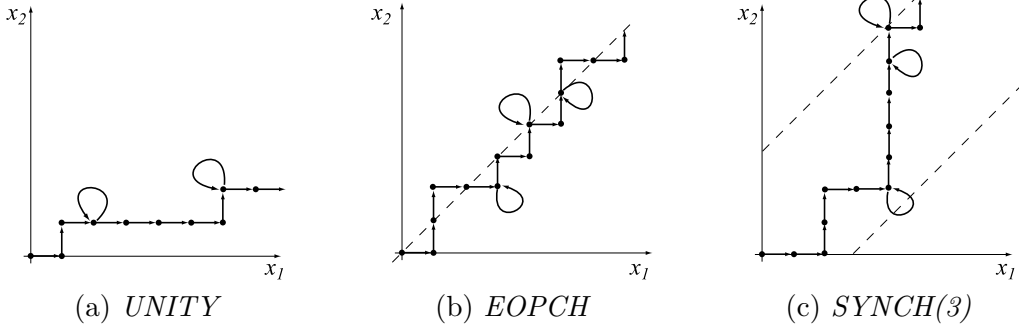


Figure 2: Three different behaviors for the two-robot example. (a) A behavior allowed by the *UNITY* semantics. The difference between x_1 and x_2 may grow arbitrarily large. The loops in the behavior indicate the occurrence of one or more stutter steps. (b) A behavior allowed by the *EPOCH* semantics. After every two non-stutter steps, $x_1 = x_2$. (c) A behavior allowed by the *SYNCH*(3) semantics. The difference between x_1 and x_2 is always less than or equal 3.

Theorem 3.1 *If P is a CCL program and F is a property, then*

- (i) $P \models_{\text{SYNCH}(\tau)} F \Rightarrow P \models_{\text{SYNCH}(\tau-1)} F$
- (ii) $P \models_{\text{SYNCH}(2)} F \Rightarrow P \models_{\text{EPOCH}} F$
- (iii) $P \models_{\text{EPOCH}} F \Rightarrow P \models_{\text{SYNCH}(1)} F$.

So if a property is true of a CCL program under a given interpretation, it is true for the more restrictive interpretation as well. This theorem along with the standard inference rules for *UNITY* [2], and other rules for reasoning about the more restrictive interpretations above, are the basis for reasoning about CCL programs in general [6].

Modeling Dynamical Systems: Unlike with HAs (discussed above), CCL programs do not make explicit use of continuous time. Also, a behavior should not be considered as defining a discrete time scale either. To make this clear, suppose we had a robot whose velocity is controlled by an external input. If we let x denote the position of the robot and u denote the commanded velocity, then the dynamics of the robot can be described by the differential equations

$$\frac{d}{dt}x = u.$$

This equation models the fact that the position is given by the integral of the commanded velocity. The solution to this equation for constant u is $x(t) = x(0) + ut$. To model this in CCL, we might write the program

$$\frac{\text{Program } P}{\begin{array}{ll} \text{Initial} & x \in \mathbb{R} \\ \text{Clauses} & \text{true} : u' = -x \\ & \text{true} : x' = x + u\delta \end{array}}$$

where δ is a small positive constant. The first command is supposed to represent the action of the robot. It senses its location x and sets the new value of u to $-x$ (just as an example). The second

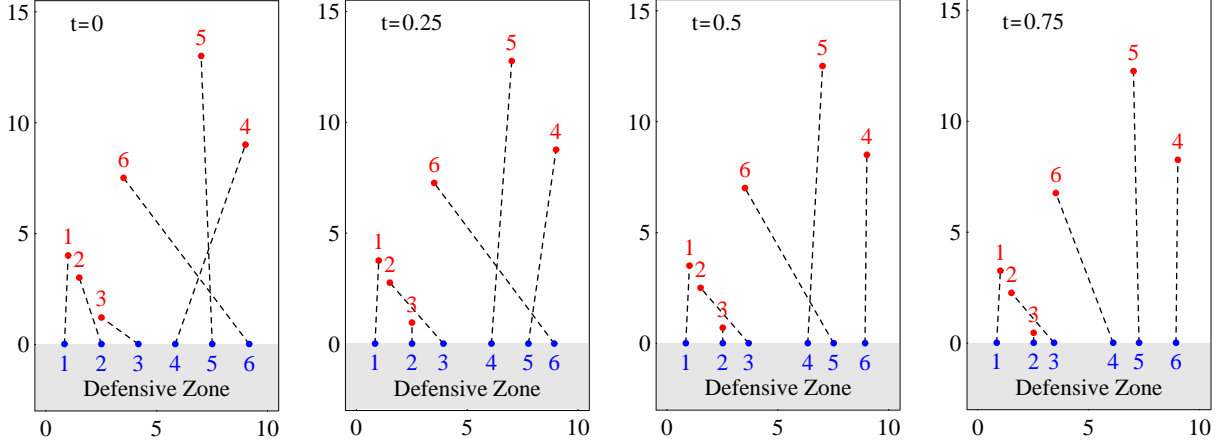


Figure 3: The first four epochs of an execution of $P_{mathitrf}(6)$. Dots along the x -axis represent blue defending robots. Other dots represent red attacking robots. Dashed lines represent the current assignment.

command is the action of the environment, which accounts for the actual motion of the robot. In the *EPOCH* semantics, for example, the first command may be executed and then the second, or *vice versa*, in each epoch. Thus it is possible that the robot executes its command twice in a row, which doesn't really do anything. It is as though the robot's sensor sent it the same value twice in a row, even though the environment (the robot's position) was changing. Only the execution of the second command by the environment accounts for any real passage of "time", measured here by the actual physical motion of the robot. If the second command happens to be executed twice in a row, it is as though 2δ seconds went by before the robot could again sense its position and act. The reader should try to convince his/herself that under the *SYNCH*(τ) semantics, the amount of "time" between each robot action varies between 1 and $\tau\delta$ seconds.

This treatment of time is a modeling choice on our part, and it is certainly subject to criticism. Our belief is that this is good enough for the problems we consider in which decentralized computation is as much an issue as physical dynamics, as the extended example in the next section illustrates. The conflict is between modeling continuous motion and modeling distributed systems, and CCL has proved, at least in our initial attempts, to strike a reasonable balance.

4 An Extended Example

We now reconsider the *RoboFlag* game discussed earlier. We do not propose to devise a strategy that addresses the full complexity of the game. Instead we examine the following very simple *drill* or exercise. Some number of blue robots with positions $(z_i, 0) \in \mathbb{R}^2$ must defend their zone $\{(x, y) \mid y \leq 0\}$ from an equal number of incoming red robots. The positions of the red robots are $(x_i, y_i) \in \mathbb{R}^2$. The situation is illustrated in Figure 3.

We first specify the very simplified dynamics of red robot i . It simply moves toward the defensive zone in small downward steps. When it reaches the defensive zone, it stays there (as there is no rule describing what to do if $y_i - \delta \leq 0$). The constants $min < max$ describe the boundaries of the

playing field and $\delta > 0$ is the magnitude of the distance a robot can move in one step.

$$\frac{\text{Program } P_{red}(i)}{\begin{array}{l} \text{Initial } x_i \in [min, max] \wedge y_i > max \\ \text{Clauses } y_i - \delta > 0 : y'_i = y_i - \delta \end{array}}$$

The motion law for the blue team is equally simple. Each blue robot i is assigned to a red robot $\alpha(i)$. In each step, blue robot i simply moves toward the robot to which it is assigned, as long as taking such an action does not lead to a collision.

$$\frac{\text{Program } P_{blue}(i)}{\begin{array}{l} \text{Initial } z_i \in [min, max] \wedge z_i < z_{i+1} \\ \text{Clauses } z_i < x_{\alpha(i)} \wedge z_i < z_{i+1} - 2\delta : z'_i = z_i + \delta \\ z_i > x_{\alpha(i)} \wedge z_i > z_{i-1} + 2\delta : z'_i = z_i - \delta \end{array}}$$

The dynamics of the entire drill system are defined by the composition

$$P_{drill}(n) = P_{red}(1) \circ \dots \circ P_{red}(n) \circ P_{blue}(1) \circ \dots \circ P_{blue}(n).$$

We now add a simple protocol for updating the assignment α . Each robot negotiates with its left and right neighbors to determine whether it should trade assignments with one of them. Switching is useful in two situations. First, if $i < j$ and $\alpha(j) < \alpha(i)$, then i and j are in conflict: intercepting their assigned red robots requires them to pass through each other. Second, if red robot $\alpha(i)$ is too close to the defensive zone for blue robot i to intercept, but not so for blue robot j , then the two robots should switch assignments. We define the predicate $switch(i, j)$ to be true if either switching the assignments of robots i and j decreases the number of red robots that can be tagged or leaves it the same and decreases the number of conflicts. The protocol is then

$$\frac{\text{Program } P_{proto}(i)}{\begin{array}{l} \text{Initial } \alpha(i) \neq \alpha(j) \text{ if } i \neq j \\ \text{Clauses } switch(i, i+1) : (\alpha(i)', \alpha(i+1)') = (\alpha(i+1), \alpha(i)) \end{array}}$$

and the full roboflag drill system is given by

$$P_{rf}(n) = P_{drill}(n) \circ P_{proto}(1) \circ \dots \circ P_{proto}(n-1).$$

Properties of P_{rf} The program we have defined has several desirable properties, which we give an overview of here. Details can be found elsewhere [6]. First, the protocol P_{rf} is *self-stabilizing* [4] in that, after an initial transient period, it settles into a mode where no assignment trades are made. That P_{rf} is self-stabilizing is expressed as

$$P_{rf}(n) \models \Diamond \Box \forall i \neg switch(i, i+1)$$

which states that it is eventually always true that no switches can be made. This is actually true under any fair interpretation of CCL programs. It can be proved using a *Lyapunov* style argument showing that at each step a certain non-negative quantity (essentially the number of conflicting assignments) must decrease if it is greater than zero. Self-stabilization is crucial in distributed computing. It states that no matter how the network is *perturbed* (e.g. by a process failure), it will eventually return to normal operation.

However, the duration of the transient is important. In particular, we desire that α stabilize *before* the red robots get too close to the defensive zone. Note that under a simple UNITY-like interpretation of the program, the red robots may move arbitrarily many times before the blue robots do, which is not our intention. Thus we can also show, roughly, that if the red robots are “far enough” away, then the blue robot’s assignments will stabilize before they arrive at the defensive zone if, for example, the *EPOCH* interpretation is used [6]. Another property that can be shown include that the blue robots never collide (fairly evident from the guards in P_{blue}).

We have thus succeeded in formally *writing down* a complete description of a multi-vehicle task, albeit a simple one, that captures how the robots move, and how they communicate with each other to achieve their objective. Furthermore, we are able to express the properties we require of the program and reason about them.

5 Programming

Because CCL has a simple, precise and formal definition, we can easily encode it in a simple programming language, which we have done. The main benefit of programming in a language like CCL is that a CCL program bears a strong resemblance to a CCL model. In fact, they might be identical. Also, the CCL style of programming is a very natural way to write programs for control systems, where often a number of threads (here represented by CCL programs) are executed in parallel (a composition of programs).

Interested readers can obtain a version of the CCL interpreter, called CCLi, at

<http://www.cs.caltech.edu/~klavins/ccl/>.

The distribution consists of the interpreter; several libraries for I/O, graphics and inter/intra process communications; a good number of examples; and a user’s manual. A CCL compiler is under construction. We describe some of the main features of CCLi here.

Expressions and Type Checking Basic CCLi expressions can be boolean, arithmetic, strings, lists and records. CCLi also provides lambda abstractions for defining functions (as in *Lisp* or *ML*) and also provides a simple mechanism for linking code written in other languages into CCLi. All expressions in CCLi are strongly typed and lists and lambda abstractions are polymorphic. CCLi performs *type inference* and *type checking* on programs before attempting to execute them, and gives useful error messages before exiting if your program is incorrectly typed.

Programs and Composition Programs in CCLi are very similar to how we have defined them above. Each program consists of a *name*, a list of parameters, a list of variable declarations and initializations, and a list of guarded commands. Variables are considered local to a program, unless they are “shared”. Thus, CCLi defines a new kind of program composition, written as follows:

$$Q(a_1, \dots, a_q) := R(b_1, \dots, b_r) + S(c_1, \dots, c_s) \text{ sharing } x_1, \dots, x_n$$

which defines a new program Q in terms of R and S in essentially the same way as standard CCL composition, except for the “sharing” part. Any variable occurring in R but not appearing in the list x_1, \dots, x_n is *local* to R in Q and similarly for S . Any variable in the list x_1, \dots, x_n appearing in R or S is considered to be the same variable. An example CCLi program illustrating composition (although not many other features) is shown in Figure 4.

```

program plant(a,b,x0,delta) := {
  x := x0;
  y := x;
  u := 0.0;
  true : {
    x := x + delta * ( a * x + b * u ),
    y := x
  }
}

program controller ( k ) := {
  y := 0.0;
  u := 0.0;
  true : { u := - k * y }
}

program system ( x0, a, b, delta ) :=
  plant ( a, b, x0, delta ) +
  controller ( 2 * a / b ) sharing u, y;

```

Figure 4: An example CCL program defining a *plant* (the system to be controlled), a *controller*, and their closed loop combination. The *controller* program can be used to simulate the system (when composed with *plant*) or executed on actual hardware if compoed with a hardware interface program (not listed).

6 Conclusion



Cooperative control presents us with the challenge of building stable control systems in a distributed control environment. To do this, we must determine at what level we want to model the systems we build so that we can ensure they have the properties we desire for them. We argued that neither a standard control theoretic approach nor a distributed systems one is completely adequate. We also reviewed several possible formalisms that seem to be appropriate for the job, finally settling on the Computation and Control Language (CCL) which seems to capture many of the essential qualities of cooperative control systems and allows us to write succinct and natural specifications and reason about their behaviors. Finally, CCL can be used to define a programming language, CCLi, that closely mirrors the CCL formalism so that our programs and models are almost one and the same.

We have found CCL useful in other situations besides reasoning about specifications. We have, for example, used CCL to express robot communication schemes so that reasoning about their *communication complexity* is straightforward [5]. In addition we have begun to explore other control related problems such as determining the state of a communications protocol (written in CCL) based on the external movements of its participants [11].

Using formalisms like CCL to do control systems design is a young endeavor and many open problems remain. For example, a main shortcoming of discrete models like CCL is that the notion of robustness to small perturbations, seeming to require a *metric* on the state space, is not well understood, much less defined. This notion is crucial to traditional control theory. Understanding this and similar problems will enable tools from control theory and distributed computation to be used in greater harmony to build the complex control networks that promise to be ubiquitous in our future.


References

- [1] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*, LNCS 736, pages 209–229. Springer-Verlag, 1993.
- [2] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
- [3] R. D’Andrea, R. M. Murray, J. A. Adams, A. T. Hayes, M. Campbell, and A. Chaudry. The RoboFlag Game. In *American Controls Conference*, 2003.
- [4] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.
- [5] E. Klavins. Communication complexity of multi-robot systems. In *Workshop on the Algorithmic Foundations of Robotics*, December 2002.
- [6] E. Klavins. A formal model of a multi-robot control and communication task. In *42nd IEEE Conference on Decision and Control*, Maui, HI, December 2003.
- [7] L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, 1994.
- [8] L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
- [9] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
- [10] N. A. Lynch, R. Segala, F. Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In *Hybrid Systems III*, LNCS 1066, pages 496–510. Springer-Verlag, 1996.
- [11] D. Del Vecchio and E. Klavins. Observation of hybrid guarded command programs. In *42nd IEEE Conference on Decision and Control*, Maui, HI, December 2003.

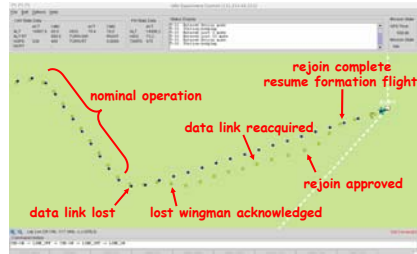



SEC Fighter/UAV Capstone Demonstration

Caltech/Colorado Team



Experiment Scenario



Scenario Timeline

- Fighter and UAV enter area near one another
- upon area entry, UAV requests rejoin & joins up
- Fighter maneuvers within agreed flight envelope _ with UAV flying in extended trail formation
- state information data link is severed
- UAV detects lost data link---initiates lost wingman
- Fighter acknowledges lost wingman
- UAV establishes approximate parallel trajectory
- UAV detects restored data link---requests rejoin
- Fighter grants permission to rejoin
- UAV rejoins & resumes formation flight maneuvering

Technologies Highlighted

Reliable wingman trajectory generation:

- online model based optimization (RHC-receding horizon control) are used to provide the UAV with *aggressive maneuvering* capabilities and the ability to work as a *reliable wingman*

Verified reliable wingman protocol:

- safe Fighter/UAV formation operations is ensured through the use of a *verified* reliable wingman protocol

Warfighting Benefits and MOEs








Online Control Customization



- UAV *autonomous maneuvering* capabilities enabled
- reliable wingman* provides ready access to UAV assets by Fighter/airborne FAC
- increased survivability & enhanced mission accomplishment

High Confidence Software Systems


- reduced *verification & validation* (V&V) costs
- guaranteed* reliable, safe autonomous operations

MOE: precise, responsive single ship and formation maneuvering without direct human control





Test Flight Planning



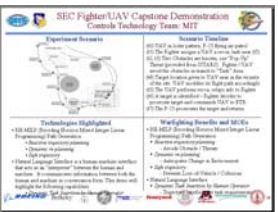
- Presentation of experiments to visitors on the ground during sorties
 - Live narration speaking to real-time displays on the Experiment Controller Display augmented with PowerPoint
 - Items to cover in presentation
 - Critical test points
 - Technology highlighted at various times in experiment
 - Warfighting benefits illustrated at various times in experiments
 - Template of PowerPoint presentation from Boeing to TDs out Tuesday 18 May 2004
 - Presentations and Narration scripts drafts due back by Friday 21 May 2004








Narration Leveraging Experiment Control Real-Time Display



Augmented with PowerPoint

- Slow periods
- Illustrative graphics describing technology



41

1